NAVAL POSTGRADUATE SCHOOL

MONTEREY, CALIFORNIA

THESIS

STRATEGIES USED IN CAPTURE-THE-FLAG EVENTS CONTRIBUTING TO TEAM PERFORMANCE

by

Wye Kede Jerel Yam

March 2016

Thesis Co-Advisors: Christopher Eagle
Robert Beverly

Approved for public release; distribution is unlimited

THIS PAGE INTENTIONALLY LEFT BLANK

REPORT DOCUMENTATION PAGE	*Form Approved OMB No. 0704–0188*

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instruction, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202–4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188) Washington DC 20503.

1. AGENCY USE ONLY *(Leave Blank)*	2. REPORT DATE March 2016	3. REPORT TYPE AND DATES COVERED Master's Thesis 09-12-2014 to 03-25-2016

4. TITLE AND SUBTITLE

STRATEGIES USED IN CAPTURE-THE-FLAG EVENTS CONTRIBUTING TO TEAM PERFORMANCE

5. FUNDING NUMBERS

6. AUTHOR(S)

Wye Kede Jerel Yam

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Naval Postgraduate School
Monterey, CA 93943

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

N/A

10. SPONSORING / MONITORING AGENCY REPORT NUMBER

11. SUPPLEMENTARY NOTES

The views expressed in this document are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government. IRB Protocol Number: N/A.

12a. DISTRIBUTION / AVAILABILITY STATEMENT

Approved for public release; distribution is unlimited

12b. DISTRIBUTION CODE

13. ABSTRACT *(maximum 200 words)*

Capture-the-flag (CTF) exercises are useful pedagogical tools and have been employed, both formally and informally, by academic institutions. Much like their physical counterparts, cyber CTF exercises hold pedagogical value and are gaining wide popularity. Existing studies on CTF exercises examined either how they benefit learning, or are best conducted. To our knowledge, no formal study has yet looked at the relationship between the strategies and tactics that the CTF participants employ (as defined by their offensive and defensive tactics), and the performance of participants in these events.

In this thesis, we studied network traffic and game state data from the DEFCON 22 CTF event. We developed tools to extract features from large volumes of network data; we then correlated these features with game state data to piece together strategies that the participating teams seemingly employ. We learned that several teams employed effective tactics such as capturing their opponents' exploits from the network to reuse them, employing automation to help with launching their exploits, obfuscating their attacks and attack responses, and attacking the client hosts of other teams.

14. SUBJECT TERMS

capture-the-flag, DEF CON CTF

15. NUMBER OF PAGES 115

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	UU

THIS PAGE INTENTIONALLY LEFT BLANK

STRATEGIES USED IN CAPTURE-THE-FLAG EVENTS CONTRIBUTING TO TEAM PERFORMANCE

Wye Kede Jerel Yam
Civilian, Ministry Of Defence, Singapore
Applied Science (B.A.Sc.), Nanyang Technological University, 2002

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 2016

Approved by: Mr. Christopher Eagle
Thesis Co-Advisor

Dr. Robert Beverly
Thesis Co-Advisor

Dr. Peter J. Denning
Chair, Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

Abstract

Capture-the-flag (CTF) exercises are useful pedagogical tools and have been employed, both formally and informally, by academic institutions. Much like their physical counterparts, cyber CTF exercises hold pedagogical value and are gaining wide popularity. Existing studies on CTF exercises examined either how they benefit learning, or are best conducted. To our knowledge, no formal study has yet looked at the relationship between the strategies and tactics that the CTF participants employ (as defined by their offensive and defensive tactics), and the performance of participants in these events.

In this thesis, we studied network traffic and game state data from the DEFCON 22 CTF event. We developed tools to extract features from large volumes of network data; we then correlated these features with game state data to piece together strategies that the participating teams seemingly employ. We learned that several teams employed effective tactics such as capturing their opponents' exploits from the network to reuse them, employing automation to help with launching their exploits, obfuscating their attacks and attack responses, and attacking the client hosts of other teams.

THIS PAGE INTENTIONALLY LEFT BLANK

Table of Contents

List of Figures

List of Tables

Acknowledgments

I am grateful for the advice and guidance provided by my thesis advisor, Mr. Christopher Eagle and co-advisor, Professor Robert Beverly. The wisdom, knowledge, and insights that they have shared with me have made this thesis a very fruitful learning experience. Thank you both for being there and spurring me on with your passion for the subject. I would also like to acknowledge and thank my wife for the support and encouragement she has given me throughout the period of my studies here at NPS.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 1: INTRODUCTION

1.1 Cyber Capture-the-Flag Exercises

Capture-the-flag (CTF) exercises are simulated combat games played by two or more opposing teams. Teams compete against one another in an attempt to capture a flag (which could be symbolic) from opposing teams, while also protecting their own flag from capture. Cyber CTFs, like their physical counterparts, involve two or more teams competing to win flags by solving cyber security related puzzles, or defending their assigned cyber infrastructure against opposing teams, while also attempting to attack their opponents' infrastructures. The popularity of CTF events has grown beyond educational institutions, and today we see CTF events organized by cyber security corporations and enthusiasts. In fact, CTF events are so popular that one is organized almost on a weekly basis.

While several papers have examined the pedagogical aspects of CTF exercises, little work has been done to examine how the game is actually played—specifically, what actions and strategies are taken by participants in these events. In this thesis, we studied network traffic and game state data from the DEF CON 22 CTF event in order to discover what strategies and tactics utilized by the participants correlated to their success in the CTF event.

1.2 Motivation

No study has examined the relationship between the strategies and tactics employed by CTF participants (as defined by the attacks and defences that they employ), and the standing of participants in these events. Such a study would prove interesting as a means to provide:

- Support for hands-on learning as a means to measure the improvement of the participants' competencies across CTF events.
- Some measure of the state of hacking techniques and tactics, as well as the evolution of the skill level of hacking groups.
- A method to determine the skill level of attackers based on network attack traffic.
- Guidance to CTF organizers in designing better competitions.

- Feedback to participants on their in-game actions so as to give them a better learning experience.
- Support to advance the state of automated network traffic analysis attack detection.

1.3 Research Questions

In this thesis, we studied the activities of DEF CON 22 CTF event participants by examining network traffic captured from the event as well as game state data made public by the event organizers [1]. We attempted to answer the following questions:

- What attack strategies do teams employ?
  - Do teams automate their attacks?
  - How fast can teams find vulnerabilities and exploit them?
  - Do teams protect their exploits against network-based detection or analysis?
  - Do teams control their rate of attack?
  - Do teams vary their attack parameters (e.g., Using different source IP addresses when attacking, or different callback ports)?
- How effective are teams at defending their vulnerable services?
  - Do teams capture exploits off the network and reuse them?
  - How fast are teams able to patch their services?
  - How effective are teams in keeping their services running?
- What are the prime factors that determine a team's final standing in the event?

1.4 Scope

For this thesis, we only analyzed the network traffic and game state data from a single CTF event, specifically the DEF CON 22 CTF. Therefore, the factors that we proposed as contributing to a team's success are only suggestions as the data set we used was not large enough for a conclusive study.

1.5 Significant Findings

We gained some insights into the strategies of participating teams. We found evidence that teams:

- Capture their opponents exploits from the network and reuse them.
- Employ automation to help with launching their exploits.
- Obfuscate their attacks and attack responses (albeit trivially).
- Attack the client hosts of other teams.

Concerning factors that correlated to a team's final score in the CTF event, we discovered that:

- **A purely offensive approach may have been more effective than an approach that balances offense and defense.** We found that the number of tokens a team loses had a low correlation to its final standing, whereas the number of tokens a team manages to steal had a high correlation to that outcome. Therefore, it stands to reason that resources put into attacking were more beneficial than resources dedicated to patching.
- **Patching was important, but not as important as keeping services up.** We saw that maintaining a high service uptime had a high correlation to a team's final standing, whereas there was a low correlation between tokens lost and a team's final standing. This implied that keeping services running, even though they were vulnerable would be more beneficial than taking them off-line to prevent them from being exploited.
- **Redeeming more flags did necessarily mean a higher final ranking.** We found that the number of tokens redeemed did not necessarily correspond to a team's final standing. This hints at some interplay between offensive and defensive factors such as tokens redeemed, tokens lost, and service uptime.
- **There was evidence to indicate that some teams had higher situational awareness than others.** Studying the scatter plots of polymorphic exploits, we saw that the top two teams switched to different exploits sometime during the CTF event. Whereas the bottom ranked team did not. As a result, the top two teams were able to exploit most of the teams for a longer duration that the bottom ranked team. This indicated that the top two teams had good awareness of what was happening and adjusted their actions accordingly.
- **Obfuscating ex-filtration traffic did not provide added benefit.** The percentage of tokens seen in the network traffic had low correlation to a team's final standing. Electing not to obfuscate ex-filtration traffic did not effect a team's final standing. In fact three of the top five teams in the competition ex-filtrated over 99% of their tokens

in the clear.

- **Exploiting a service multiple times per round had no added benefit.** We saw that most teams exploited each service multiple times per round, even though exploiting a service once successfully was sufficient to steal a token. From the Pearson coefficient scores, we saw little correlation between how many exploits were launched per round versus how the teams scored in the end.
- **The method of ex-filtration used did not matter.** We looked at two payloads that the teams used—session-reuse payloads and callback payloads. Comparing teams that used only one type of payload with teams that used both, we did not find any correlation between the type of ex-filtration techniques used, and a team's ranking. We found half of the teams used both types of payloads and the other half used only session-reuse payloads.

1.6 Thesis Structure

The remainder of the thesis is structured as follows:

1. Chapter 2 covers work related to studying CTFs. It also covers tools and techniques used in the processing, visualizing, and analyzing of network traffic captures.
2. Chapter 3 covers the methodology and processes used to study the network traffic captures and data from the scoring database.
3. Chapter 4 covers the analysis of the results and insights that were learned.
4. Chapter 5 concludes the study and recommends areas of future work.

CHAPTER 2:
BACKGROUND AND RELATED WORK

Previous studies on CTFs have generally fallen into two categories. The first category deals with the utility of CTFs. It is generally accepted that CTF games are useful pedagogical tools. For example, Dabrowski showed that using CTF-like challenges as part of a security class motivated students to put more effort into their learning [2]. Furthermore, Carlisle shared that incorporating CTFs into a curriculum at the United States Air Force Academy led to increased student interest in cyber security studies, greater collaboration among students, and greater willingness towards self-directed studies [3]. Chothia also found a correlation between students doing well in jeopardy-style CTFs and them performing well in more formal assessments [4].

The second category deals more with the mechanics of organizing and running a CTF, and generally includes lessons learned from past CTFs and ideas on how to improve those conducted in future. For example, Chung described the shortcomings and strengths of various CTF events in terms of game design and their pedagogical efficacy [5], and both Davis [6] and Vigna [7], respectively, wrote about the architecture of the MIT/LL CTF and iCTF as well as their experiences in organizing these CTF events.

While participating teams do narrate their experiences in various CTF events (such as [8], [9] and [10]), we know of no systematic study that has been done on how participants play the CTF game. Nor do we know of any systematic studies that examine the relationship between the tactics of the CTF participants and their corresponding final rankings. This thesis examined game state data as well as network activities of the DEF CON 22 CTF event participants. The goal of this thesis was to in an attempt to discover their strategies, and derive relationships between the offensive and defensive capabilities of the teams and their eventual ranking in the event.

2.1 Types of CTF Events

CTF events are cyber security competitions where participating teams compete with one another to capture virtual flags. Depending of the type of CTF event, flags can be captured either by solving cyber security-related puzzles or by compromising the cyber infrastructures of opposing teams. According to CTFTime.org [11], there are three types of CTF events:

1. **Jeopardy-style.** These CTF events have various categories of challenges that are linked in a chain. As teams solve the challenges, more difficult challenges higher up in the chain are made available. Teams are awarded flags or points for each challenge solved and are awarded more points for solving tougher challenges. The usual categories in jeopardy-style CTFs are:
   - Pwnables. These challenges require teams to find and exploit a vulnerability in programs written by the organizers.
   - Crypto. These challenges require teams to break a custom encryption algorithm or decode a given cipher text.
   - Reverse Engineering. These challenges require teams to reverse engineer programs written by the organizers to understand how they work and recover a flag from program.
   - Web. These are web-based challenges that require teams to exploit a web application to gain access to a file containing the flag.
2. **Attack-Defense.** In these CTF events, each team is given a network (or a single host) with several vulnerable services that they must defend. Teams defend their services by reverse engineering the services to understand their functionality, find their vulnerabilities and then patch the vulnerabilities. In addition to defending their vulnerable services, teams also attempt to compromise their opponents' vulnerable services to access a specific file on the target system (which is akin to capturing a flag). Teams gain points for capturing flags and have points deducted if their services go down or if their flags are stolen.
3. **Mixed CTF.** These CTFs come in a variety of formats and could be similar to an attack-defense CTF but with the incorporation of task-based elements.

2.2 Elements of an Attack-Defense CTF

In most Attack-Defense type CTFs, there are the following elements:

- **Real-Time Scoreboard.** The scoreboard keeps track of the scores or the number of flags captured by each team. This scoreboard is normally connected to the game scoring server (to which teams submit their captured flags) so that the scoreboard is updated in real-time whenever a team submits a flag.

- **Rounds of play.** A CTF event is normally split into rounds of equal duration. During each round, the flags of each vulnerable service are changed (i.e., the contents of the flag files on the servers are changed). Hence, flags captured in one round are no longer be valid in subsequent rounds and, teams have to exploit their opponents services in every subsequent round to capture new flags.
- **Network Infrastructure.** Every team is given a server running a number of vulnerable services that they must defend. For CTF events that are held at a single site(e.g., DEF CON CTFs), these servers are all located on the same network. For CTF events that are multi-site (e.g., the UCSB iCTF), teams are connected to the rest of CTF infrastructure via a VPN tunnel. In these multi-site CTF events, the server containing the vulnerable services may be distributed to the teams ahead of time via an encrypted tarball. On the day of the CTF event, the encryption key for the tarball is given to the teams. The teams then decrypt the server image and host it on their own virtual machine infrastructure.
- **Exploitation Mechanics.** In some CTF events, participating teams are directly connected to one another on the network. Hence, teams can directly attack other servers from their own hosts. In some CTFs (e.g., UCSB iCTF), teams are not directly connected and cannot attack other teams from their own hosts. Instead, teams wishing to attack other teams must submit their exploits to the organizers, who then launch the exploits on behalf of the teams.

2.3 CTF Strategies

There are many strategies and tactics that participating teams employ to gain an advantage over other teams. Some of the these strategies are:

- **Reusing Exploits.** Rather than developing its own exploits, a team can choose to capture the exploits used against them and use them to attack other teams. The challenge here is detecting the exploit on the network.
- **Detecting Exploits.** Some teams use an IDS in an attempt to detect attacks launched against them. However, since the exploits launched against them will be custom written, there are unlikely no IDS signatures for these exploits. Hence, using an IDS

to find exploits on the network may not be a fruitful endeavour.

- **Detecting Outgoing Flags**. Since detecting attacks using IDSes may not always be possible, teams may look instead for flags being ex-filtrated from their servers. Since ex-filtrated flags indicate compromise, looking for flags instead of the exploits themselves is a feasible method for detecting an attack. In some CTF events, flags have a fixed format and are therefore easy to spot on the network.
- **Obfuscating exploits.** In order to prevent other teams from capturing and reusing their exploits, teams can obfuscate their exploits to make it more difficult to analyze them. Teams may also use multi-stage exploits to make it more difficult for their exploits to be pieced together as a whole.
- **Obfuscating ex-filtration traffic**. In addition to obfuscating their exploits, teams may obfuscate their ex-filtration traffic so that the flags they capture are not detected on the network, which increases the likelihood that their attacks go unnoticed.
- **Mangling flags.** One of the best defense a team can employ to protect their services from exploitation is to patch them. However, there is often more than one vulnerability in a service, and therefore no conclusive way to know that all vulnerabilities have been patched. Knowing this, teams may implement code to modify the flag as it leaves their server. This achieves the effect of frustrating the team that captured the flag because the flag would be invalid and the game server would reject it when the mangled flag is submitted.
- **Attacking clients.** Instead of attacking the servers of their opponents, teams can choose to attack their opponents clients. Successfully exploiting their opponents clients may give teams an opportunity to disrupt their opponents' activities and steal their exploits.
- **Collusion.** Another strategy that teams may choose adopt is to work with other teams in a mutually beneficial way. Some examples of collusion would be sharing exploits or tokens.
- **Leverage on external resources.** Teams may leverage computing resources outside of the CTF network to perform computing intensive tasks such as brute forcing keys. Teams may even set up Internet Relay Chat rooms to enlist the help of fellow hackers not officially participating in the CTF to help with solving the challenges.

2.4 Tools Used in CTFs

The most common tools used by CTF participants are:

- **Disassemblers and Debuggers.** Teams use disassemblers to disassemble the services into assembly code in order to statically analyze the services and find their vulnerabilities. Teams also use debuggers to dynamically analyze the runtime behaviour of the services.
- **Exploitation Frameworks.** Teams may use exploitation frameworks to help them develop exploits or payloads for the vulnerabilities they have discovered. A common exploitation framework is the Metasploit Framework, which is a publicly available collection of exploits and shellcodes. The exploits in the Metasploit Framework are for publicly available software, and therefore will not work for the services in CTFs. However, the collection of shellcodes in the Metasploit Framework can be used for the services.
- **Scripting languages**. Teams need to write code to automate the launching of their exploits. Scripting languages such as Python or Perl can be useful for this purpose.
- **Network traffic analysis tools**. Recall that one of the strategies employed in CTFs is to re-use exploits captured from other teams. Teams use network traffic analysis tools, like Wireshark, to capture network traffic going to and leaving their network to search for exploits and/or ex-filtrated tokens.

2.5 DEF CON CTF

DEF CON is an annual cyber security conference held in Las Vegas, NV. According to its official website, DEF CON was started in 1993 as a party for hacking enthusiasts, all of whom were part of an electronic bulletin board service network [12]. In 1996, DEF CON began holding formal annual CTF competitions, though prior to this, CTFs were also held, albeit in a less formal capacity. Since then, the DEF CON CTF has grown to be the most prestigious of all CTF events, and dubbed by CNBC as the “World Series of hacking” [13]. The DEF CON 22 CTF event was the 19th CTF event to be held at DEF CON. It was a three day event held August 8-10, 2014.

The DEF CON CTF events throughout the years have all been attack-defense type CTFs. While this has not changed since its inception, the organizers, the scoring rules, the types of services, operating systems, and architectures have varied annually. The number of vulnerable services for each iteration of DEF CON CTF can range from five to about twenty. According to DEF CON CTF history by Dark Tangent [14], the services can range from "poorly configured crypto, SQL-injection, cross-site-scripting, buffer overflows, timing attacks, heap exploits, malformed network constructs and custom interpreters." The operating systems are usually Linux-based (with FreeBSD being used on some occasions) and the architectures vary widely between x86, x64, ARM, and embedded systems.

For our thesis, we chose to study DEF CON 22 CTF because it was a recent CTF, and so the strategies and tactics employed by its participants would be current and relevant.

2.5.1 DEF CON 22 CTF Finalists

There were twenty teams in the DEF CON 22 CTF finals. According to [15], these teams came from diverse nations such as the United States (*ppp, shellphish*), Taiwan (*hitcon*), South Korea (*raon_asrt, GoN, codered*), China (*blue-lotus*), Russia (*more smoked leet chicken, balalaikacr3w*), Australia (*9447*), Germany (*StratumAuhuur*), Japan (*binja*), France (*w3stormz*), Denmark (*gallopsled*) and Poland (*dragonsector*). The largest representation come from South Korea (3 teams) followed by the United States (2 teams) and Russia (2 teams). We could not find information about the nationality of five teams: *reckless abandon*, *routards*, *penthackon*, *hackingforchimac*, and *(mostly) men in black hats*.

A team may qualify for the DEF CON CTF finals in three ways. The first way is to take part in, and win, one of several qualifying CTF events. These qualifying events are usually jeopardy-style CTFs organized by cyber security interest groups. Examples of such CTF events include PlaidCTF [16], Boston Key Party [17] and RuCTFE [18]. The second way to qualify for the finals is to take part in the official DEF CON CTF qualifiers and finish as one of the top ten teams. The third way is to return as defending champion of the previous year's DEF CON CTF event. Table 2.1 shows the finalists for the DEF CON 22 CTF as well as their respective qualifying events.

Table 2.1: Table of DEF CON 22 CTF finalist and their respective qualifiers

Team(s)	Qualifying Event
Plaid Parliament of Pwning (PPP)	DEF CON 21 CTF Winner
StratumAuhuur	Boston Key Party
More Smoked Leet Chicken (mslc)	RuCTFe
Dragon Sector	Ghost In The Shellcode
[SEWorks]penthackon	Olympic CTF
Gallopsled	Codegate 2014 Finals
BalalaikaCr3w	PHDays 2014
Binja	SecuInside 2014
9447, Reckless Abandon, Routards, Raon_ASRT, KAIST GoN, shellphish, CodeRed, HITCON, blue-lotus, HackingForChiMac, (Mostly) Men in Black Hats, w3stormz	DEF CON 22 CTF Qualifier

Finalist information compiled from [19]

2.5.2 DEF CON 22 CTF Game Infrastructure

Every iteration of the DEF CON CTF event is unique and the corresponding operating systems, architecture, and type of vulnerable services change from year to year. In the DEF CON 22 CTF event, every team was given a single host running four vulnerable services. Each team was responsible for defending its assigned host against attacks from opposing teams. As described by Stratum [20], the host was running on an ARM-based ODROID-U2 development platform, and the four vulnerable services were as follows:

1. eliza – A text-based space economy simulator
2. wdub – A web service
3. justify – A constraint solver
4. imap – An IMAP email server

In addition to the four vulnerable services, there was also a separate hardware challenge called Badger. Badger was an electronic circuit board that ran a radio chat service, and teams had to exploit the firmware running on the opposing team's circuit boards via the radio chat service. There was also a sixth vulnerable service that the organizers created called csniff, but it was never put into play during the event.

2.5.3 DEF CON 22 CTF Team IP Assignments

Each team in the DEF CON 22 CTF event is given a CIDR subnet and each team's server, that hosts the vulnerable services, is given a .2 IP within the subnet. For example, if team PPP is given the subnet of 10.5.1.0/24, then PPP's server will have the IP 10.5.1.2. As a result of the IP assignments, we were able to identify traffic to/from a team's server or client by the IP address alone. Table 2.2 gives the IP subnets assigned to each team in the DEF CON 22 CTF event.

Table 2.2: Table of Team to IP subnet mappings

Team	Assigned CIDR subnet
Plaid Parliament of Pwning (PPP)	10.5.1.0/24
9447	10.5.2.0/24
Reckless Abandon	10.5.3.0/24
Routards	10.5.4.0/24
raon_ASRT	10.5.5.0/24
KAIST GoN	10.5.6.0/24
shellphish	10.5.7.0/24
CodeRed	10.5.8.0/24
HITCON	10.5.9.0/24
blue-lotus	10.5.10.0/24
HackingForChiMac	10.5.11.0/24
(Mostly) Men in Black Hats (mmibh)	10.5.12.0/24
w3stormz	10.5.13.0/24
More Smoked Leet Chicken (mslc)	10.5.14.0/24
Dragon Sector	10.5.15.0/24
[SEWorks]penthackon	10.5.16.0/24
Stratum Auhuur	10.5.17.0/24
Gallopsled	10.5.18.0/24
BalalaikaCr3w	10.5.19.0/24
binja	10.5.20.0/24

2.5.4 Game Mechanics and Scoring

The DEF CON 22 CTF event was broken up into 272 rounds that each lasted five minutes. At the start of the CTF event, each team was assigned 2,502 flags. These 2,502 flags were further distributed equally among the six services (even the sixth, disabled, service was assigned flags), with each service being assigned 417 flags. Ultimately, the number of flags a team owned (be it flags they captured or flags that belonged to them from the start) determined their score. The team with the greatest number of flags at the end of the event had the highest score.

In every round of the event, teams captured flags by exploiting the vulnerable services of opposing teams, stealing a token, and redeeming the token with the scoring server. By redeeming a token, a team indicated to the scoring server that it had successfully exploited an opposing team's service. So, at the end of a round, flags were distributed to teams that redeemed tokens for that round. A token was a random string of letters and digits (e.g., zXKTilzOlxsnYZvsZqjsxnSDXi) that was associated with a specific service and a specific round. Exploiting an opponent's service allowed a team to read a file that contained the token. All stolen tokens could be redeemed for points. A new token was generated for each service every round; hence, teams needed to exploit their opponent's services every round in order to steal a valid token for a particular round.

At the start of each round, each team's service had a maximum of 19 points available for redemption (as long as there were points available). At the end of every round, all teams that had successfully compromised a service and redeemed its associated token was given a share of these 19 points. A team that had its service exploited lost all 19 points associated with that service, independent of the number of opposing teams that exploited their service in that round. Points received from one service were credited to the corresponding service of the capturing team. For example, in one round, if five teams successfully exploited and redeemed the token associated with the Eliza service of team PPP, then at the end of the round, PPP would lose the 19 points that were associated with Eliza and the five teams would distribute the 19 points equally among themselves, with each getting 3 flags (the 3 flags were derived from dividing the 19 points by the 3 teams and rounding down). The remainder of the points that were not distributed would be acquired by the organizer. The points that each of the five teams received would be added to the point total of their respective Eliza service. In addition, if a team's service was down for a round, that team also would lose the 19 points associated with its service and these 19 points would be distributed between the teams that have that particular service up.

2.5.5 Network Data Organization

The organizers of the DEF CON 22 CTF made the captured network traffic of the event available online. It consisted of 285GB of full packet network traffic split according to teams. The 285GB of captured network traffic was saved in the libpcap file format and split across multiple files, with each file (called pcap files) containing network traffic captured within a 5-minute window (some pcap files contain network traffic lasting over a longer duration). Depending on how much data was sent over the network in a 5-minute window, the size of each pcap file ranged from 100KB to well over 100MB.

Although each pcap file contained network traffic captured roughly within a 5-minute duration, these five minutes did not necessarily fall within the window of a round in the CTF event. Each file potentially contained network traffic that straddled two consecutive rounds and this fact was accounted for when we inspected round-by-round network traffic. Figure 2.1 illustrates how the network traffic files corresponded to round boundaries.

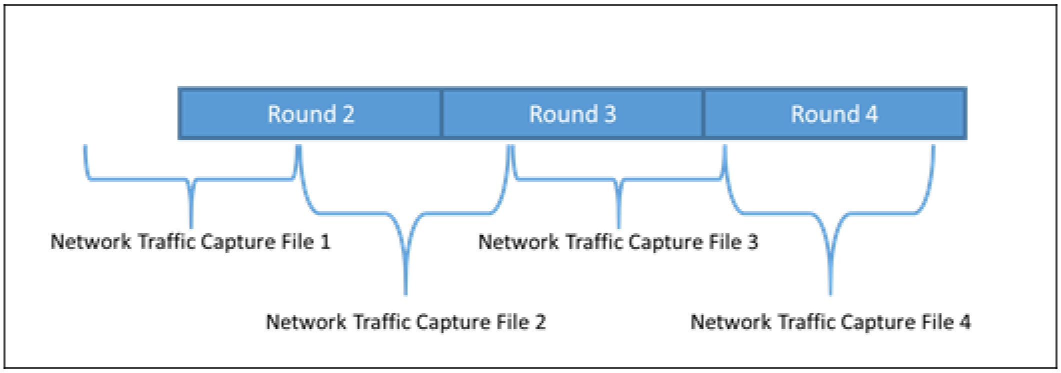

Round 2
Round 3
Round 4
Network Traffic Capture File 1
Network Traffic Capture File 3
Network Traffic Capture File 2
Network Traffic Capture File 4

Figure 2.1: Illustration showing how files of captured network traffic may overlap round boundaries

Each team's set of pcap files should have contained all network traffic going to and from their hosts. Hence, if a team attacked another, we expected to see their attack traffic appear in the pcap files of both these teams (albeit with the source and destinations reversed). However, we noticed that this was not the case and we detected a number of instances where traffic appearing in one team's pcap file was not present in the file of the other. We do not know for sure the reason for this discrepancy, but it may be due to the packet capturing system being overwhelmed and ignoring packets. Hence, when searching through the pcap files for tokens, we had to be mindful of this fact and searched the pcap files of both teams involved in the network conversation.

Not all traffic going to and coming from the vulnerable services was captured in these pcap files, however, as one of the services, Badger, worked over a radio chat service and therefore did not send data over the network where the organizers were capturing the traffic. As a result, our analysis was missing all data pertaining to the Badger service. Fortunately, based on anecdotal evidence from *routard's* write-up [21], we found that little data was exchanged over the Badger radio service as Badger challenging and only *routards* was reported to have successfully compromised this service. Hence, we do not expect that missing data from the Badger challenge affected our findings.

2.5.6 Game State Data

The game state data for the DEF CON 22 CTF contained information on the state of the CTF event at every round. All data was stored in a series of PostgresSQL database tables. The game state information that was stored in the database included:

- The tokens redeemed each round, the team redeeming the tokens as well as the team from which the token was stolen.
- The start time and end time of each round.
- The number of flags captured for each round (as a result of flag redistribution based on tokens redeemed).
- The rounds in which services were down.

While the game state data stored only the final scores, it was possible to calculate the scores for any one round from the data in the database. The entity relationship diagram, which illustrates how the tables of the diagram relate to one another, is shown in Figure 2.2.

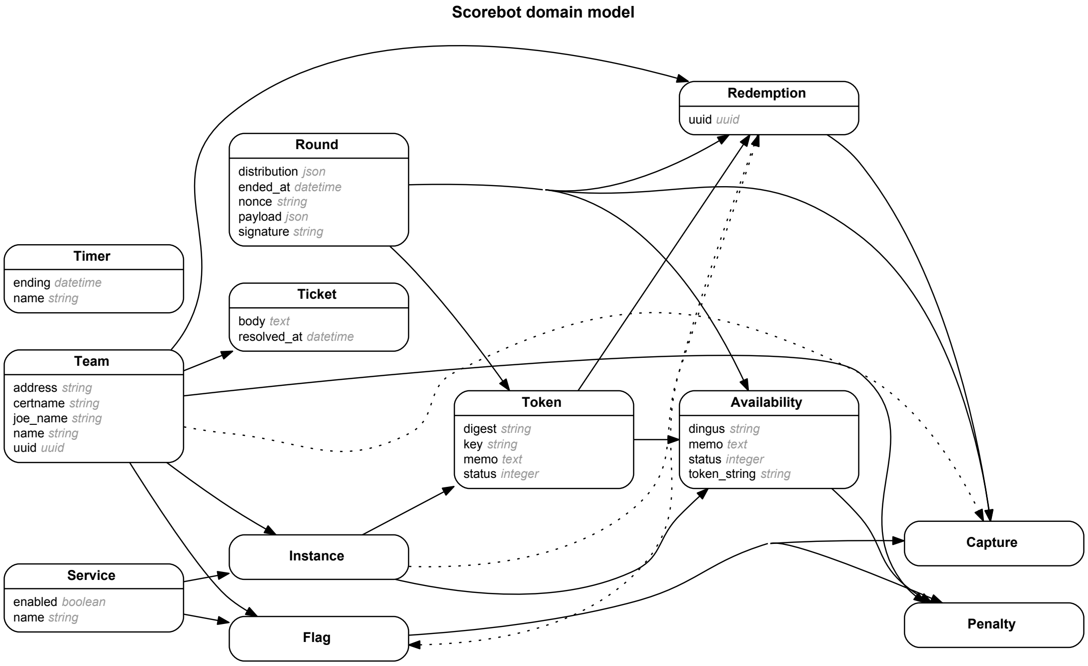


Figure 2.2: Entity Relationship Diagram of game state database

ERD diagram taken from LegitBS “ScoreBot Domain Model," 2014 [1]

2.5.7 DEF CON 22 CTF Visualization

The organizers created a real-time visualization to update the the participants and spectators on the progress of the CTF event. As described in [22], the visualization had boxes that represented each team, and their physical position within the room. Every successful attack by a team was displayed as projectiles launching from the attacking team to the exploited team. The visualization for the DEF CON 22 CTF can be found at https://www.youtube.com/watch?v=1UT3qXHduts&feature=youtu.be

2.6 Network Flows

In this thesis, we made use of network flows to analyze trends in network traffic. As defined in SiLK FAQ [23], Network flow is a summary of a network conversation that covers all traffic matching five attributes that are relevant for addressing: "the source and destination addresses, the source and destination ports, and the protocol. These attributes (sometimes called the 5-tuple), together with the start time of each network flow, distinguish network flows from each other."

2.7 Network Traffic Analysis Tools

A major part of this thesis focused analyzing network traffic captures of the DEF CON 22 CTF event. Given the amount of network activity, it would have been infeasible to manually process and inspect. Hence, we used the following tools to aid our analysis.

2.7.1 SiLK

The System for Internet-Level Knowledge (SiLK) is termed as "a collection of traffic analysis tools developed by the CERT Network Situational Awareness Team (CERT NetSA) to facilitate security analysis of large networks" [24]. SiLK utilizes its own space-efficient binary flat-file format to store network flow information and has a set of analysis tools that can be used to perform queries on this file format. The analysis tools interoperate using pipes, which made it convenient for us to chain them together to perform complex analyses. Specifically, we used the following tools in SiLK:

- **rwp2yaf2silk.** This tool reads the content of one or more pcap (libpcap library format) files and coverts the data into SiLK flow records. It creates a flow record from associated packets and is also able reassemble fragments into packets prior to conversion.
- **rwfilter.** This tool filters flows from SiLK flow data based on properties such source or destination IPs and ports. It can be used to read SiLK flow records from stdin and send its output to stdout. As a result, several instances of rwfilter can be chained together to create complex filtering rules.
- **rwcut.** This tool essentially prints out SiLK flow data in human readable format. It is generally used as an interface to pipe SiLK flow records to other programs that do not accept SiLK flow records.

2.7.2 nGrep

ngrep is a tool written by Jordan Ritter and used to analyze network packets [25]. It is mainly used to search for byte patterns within pcap files (or live network traffic) by specifying regular expressions. Like tcpdump, it accepts the Berkeley Packet Filter (BPF) syntax for specifying filters to select only specific traffic within which to search.

2.7.3 tcpflow

tcpflow is touted as a TCP/IP session assembler. Supplied with pcap files (or live network traffic), tcpflow will reassemble packets belonging to the same flow, extract the payload, and save each flow's payload into a separate file.

2.8 Network Visualization Tools

Often, having a good visual representation of the network data will aid in its analysis. For example, being able to visualize network traffic trends across time could provide hints about the work tempo of each team, or being able to see who the hosts on the network are talking to could reveal anomalous conversations. We used an ensemble of the following tools to help us visualize the DEF CON 22 CTF network traffic.

2.8.1 Google Charts

Google Charts is a toolbox for drawing interactive charts, such as line charts and pie charts, and is a convenient way for visualizing data. It does not require 3rd party plugins and is based on JavaScript and HTML 5 [26], which are supported by all modern web browsers, making it convenient to use.

2.8.2 Flow Plotter

FlowPlotter is a tool developed by Jason Smith that reads in SiLK flow data and generates visualizations such as line charts, bubble charts, and directed graphs [27]. FlowPlotter uses Google and D3 charts to generate its visualizations. We used FlowPlotter to visualize the DEF CON 22 CTF network traffic and were able to spot interesting network trends that provided hints as to how teams used automation as part of their attack methodologies.

2.8.3 Gephi

Gephi is open source software for visualizing and understanding graphs and networks [28]. It contains functionality for visualizing and interacting with network and graph data, and it utilizes a plugin framework so that new functionality can be added to Gephi. By using the zoom and pan functions, we were able to spot suspicious connections between hosts.

CHAPTER 3: METHODOLOGY

While it may be tempting to treat the analysis of DEF CON 22 CTF data like that of network intrusions (since both deal with studying malicious network traffic), and therefore adopt a similar methodology, there are in fact fundamental differences between the two. The key differences being:

- The majority of the traffic in DEF CON 22 CTF was malicious traffic, whereas in a normal network, the bulk is comprised of benign traffic.
- The networks respective definitions of anomalous traffic are different. In regular networks, anomalous traffic often exhibits adversarial behaviour. In the DEF CON 22 CTF traffic, almost all traffic we expected to see was adversarial and therefore anomalous by normal standards. Hence, we needed a different set of considerations to differentiate traffic that is considered anomalous by DEF CON CTF standards.
- Exploits used in normal networks target known services and have known signatures. In the DEF CON 22 CTF event, the services were custom written for the competition. Therefore, the exploits that the participants develop for use against them have no available public signatures.

Bearing in mind the above differences and the fact that we had a tremendous amount of network data to sift through, we found that identifying interesting network packets by relying on signatures or performing an exhaustive search was not possible. Therefore, we adopted the following methodology:

1. We processed the network traffic (that contains the full payload) into a light-weight representation so that we could more effectively visualize and summarize.
2. We used data visualization techniques to help zoom in to areas of interest.
3. We correlated network data with game state data as well as with source code of scoring infrastructure to efficiently narrow down the network traffic of interest.
4. We adopted the approach of using one successful exploit from each team as representative for every instance of a team's exploit. This saved us the work of having to inspect every instance of every exploit from each team, and reduced it to inspecting

one exploit instance for every exploit from each team. For example, throughout the CTF event, team *ppp* launched 1000 instances of its exploit against the Eliza service. Instead of studying all 1000 instances, we assumed that all 1000 instances of the exploit are identical, or nearly so, and we studied only one of the instances.

The sources of data used in our analysis are as follows:

1. Network Traffic of the DEF CON 22 CTF competition
2. Game state data from the scoring database
3. Final scores of the teams

3.1 Processing Network Traffic Data

The DEF CON 22 CTF network traffic that we obtained were full packet captures, which contained not only packet headers, but also the payload. In addition to being large, they also potentially contained traffic that did not directly relate to our analysis. For example, the packet captures contained a lot of Internet-bound traffic that were erroneously routed into the game network. There were also a lot of multicast packets that were sent by services running on the team's client machines that were not intentional attack traffic. Therefore, we needed to convert the packets to a representation that was more efficient in terms of storage requirements while remaining suitable for statistical analysis. For this purpose, we chose the SiLK file format as the format in which to save the network traffic flow information.

The organizers split the network traffic of each team into 306 pcap files. With 20 teams and 306 pcap files per team, we had a total of 6120 pcap files to process. Such a large number of pcap files made the analysis cumbersome. For performing network traffic trends analysis, we did not require payload information; hence, we used the rwp2yaf2silk script to extract flow records from the 306 pcap files of each team. We then saved the extracted flow records into 20 SiLK files—one SiLK file for each team—thereby greatly reducing the number of files we had to process.

Using SiLK flow records, we greatly sped up the analysis of areas that required only flow record information. For example, in studying the number of hosts each team used to launch their attacks and the number of callback ports each team used for their reverse connect payloads, we used flow record data. For other analysis that required extraction of tokens and exploits, we used the processes described in section 3.4, 3.5, and 3.6. The process of converting the pcap files to SilK flow records is illustrated in Figure 3.1.

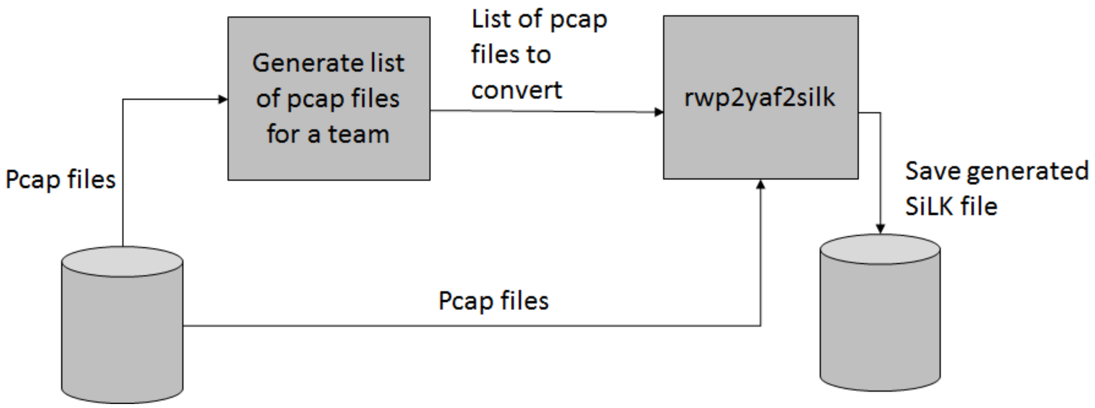

Generate list of pcap files for a team
List of pcap files to convert
rwp2yaf2silk
Pcap files
Save generated SiLK file
Pcap files

Figure 3.1: Process of converting a team's pcap files to SiLK flow records file

3.2 Visualization of SiLK Flow Records

Often visualizing information allows one to come up with insights that would otherwise be impossible. Especially when it comes to searching for anomalies, visualizing data may be a helpful technique. For example, we wanted know whether teams automate their attacks. By visualizing network traffic trends, we could spot possible instances of automation from the presence of regular spikes in network activity. In addition, we also wanted to know whether teams attempted to exploit the clients of other teams. This could be determined by observing a network graph that captured host-to-host conversations and looking for instances where clients of one team connected to clients of another team . Hence, we used two visualization tools to help visualize the DEF CON 22 CTF network traffic. The tools are flowplotter, which we used to visualize network traffic trends, and Gephi, which we used to visualize conversations between hosts.

As we had already extracted network flow information from the pcap files into SiLK flow records, we could easily use flowplotter and Gephi to visualize those flow records. Figure 3.2 shows the process of using the SiLK toolkit with flowplotter to generate network traffic trends so that we can visually spot regular occurring network spikes. We first read in the SiLK flow records of each team and used rwfilter to filter non CTF-related traffic (such as Internet-bound packets that were erroneously routed into the CTF network and multicast service discovery related packets). We then piped the rwfilter output to flowplotter and then combined all of their individual line charts into one single line chart.

Figure 3.3 shows how we use the SiLK toolkit with Gephi to visualize host-to-host conversations so that we could spot conversations that only involved clients. Similar to processing the SiLK flow records for visualizing network traffic trends, we read in the SilK flow records for each team, and used rwfilter in series to select only network conversations that involved client hosts and filtered away non CTF-related traffic. We then piped the output of rwfilter to rwcut to convert the filtered SiLK flow records into a textual form. Finally, we used sed to covert the output of rwcut to a dot file format, which we then displayed with Gephi.

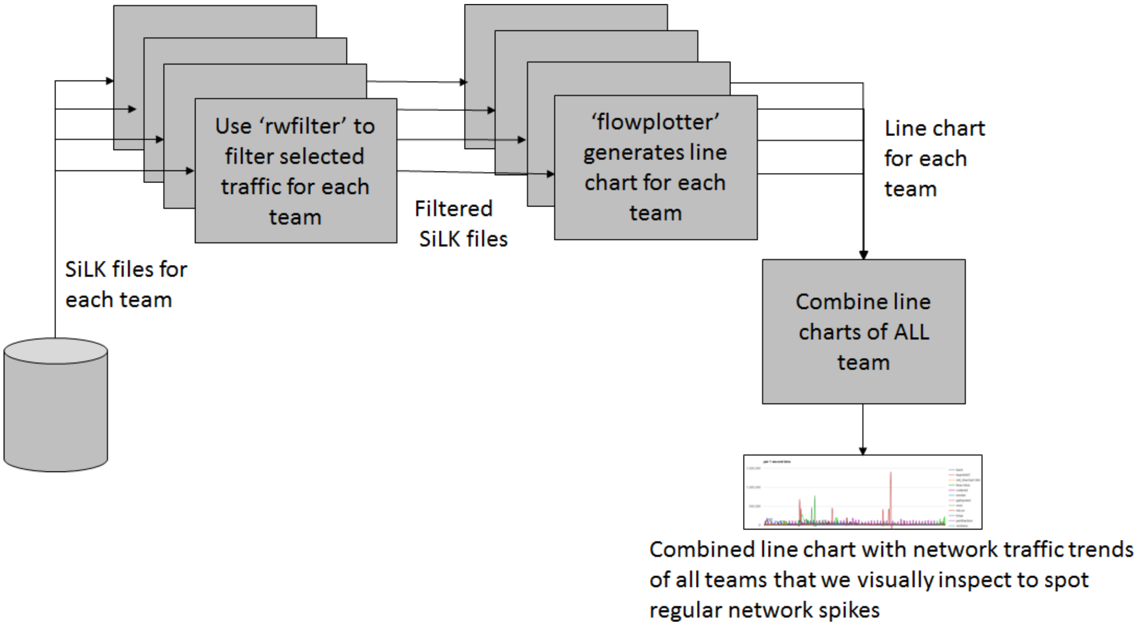

Use 'rwfilter' to filter selected traffic for each team
Filtered SiLK files
'flowplotter' generates line chart for each team
Line chart for each team
SiLK files for each team
Combine line charts of ALL team
Combined line chart with network traffic trends of all teams that we visually inspect to spot regular network spikes

Figure 3.2: Process to generate and visualize network traffic trends

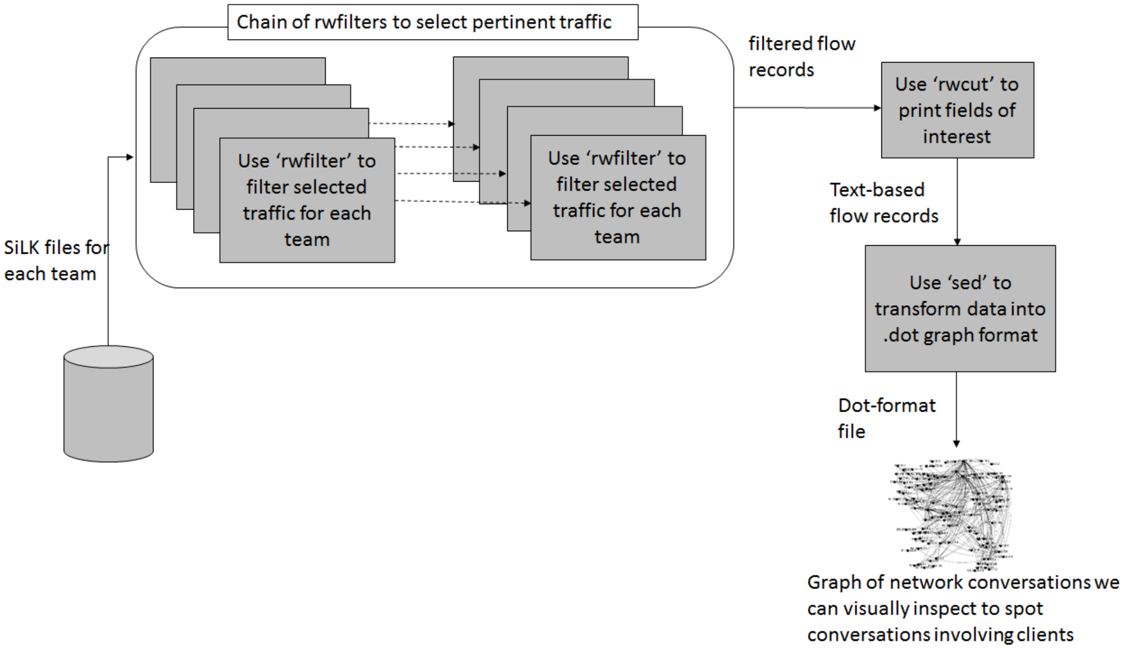

Chain of rwfilters to select pertinent traffic
Use 'rwfilter' to filter selected traffic for each team
Use 'rwfilter' to filter selected traffic for each team
SiLK files for each team
filtered flow records
Use 'rwcut' to print fields of interest
Text-based flow records
Use 'sed' to transform data into .dot graph format
Dot-format file
Graph of network conversations we can visually inspect to spot conversations involving clients

Figure 3.3: Process to generate and visualize network host-to-host conversations

3.3 Correlating Game State Data and Network Traffic to Narrow Search Results

While visualizing network conversations and trends may have provided interesting insights, we needed to go deeper and actually inspect the payloads of the network packets in order to gain a better understanding of the offensive techniques employed by the teams. Specifically, we were interested in knowing if:

1. Teams obfuscated their exploits to minimize the presence of attack indicators.
2. Teams obfuscated their ex-filtration traffic to hosts they compromised in order to minimize indicators of compromise.
3. Teams used public shellcodes, like those found in the Metasploit Framework or those found on public shellcode repositories.
4. Teams limited the rate of exploitation.

However, there were two challenges to more deeply inspecting the networks' traffic:

1. There was too much network traffic to manually sort through in a reasonable amount of time.
2. We were initially unable to extract exploits automatically because we did not have any reference data to use as signatures. In addition, tokens were random strings of digits and letters of 26 characters in length and using a general regular expression for matching against tokens would yield a large number of false positives.

In order to overcome these two challenges, we used game state data to help narrow the search space. From the game state data, we were able to:

- **Discover the rounds (and therefore the time window) within which tokens were redeemed.** Knowing the time window gave an approximate time when a service was attacked, since a team must have successfully compromised a service before stealing a token. In addition, knowing the time window in which a token was redeemed also narrowed the search space of network traffic in which to look for stolen tokens, an absence of which indicated obfuscated tokens.
- **Discover the partial token which we used as a search pattern to automatically find the tokens in network traffic.** Unfortunately, the game state database did not store the full token, rather, it stored every other character of the token. For example, if a token was "1234ABCD," then the information stored in the database would only be "13AC." Presumably, this measure was taken for the sake of security, so that if the database was compromised, the whole token was not revealed. The hash of the whole token was stored in the database; hence, when a team redeemed a token, the scoring system could verify that the token was indeed valid. Consequently, we were not able to do a simple pattern search of the network traffic to look for partially known tokens, but instead used regular expression and verified the tokens we found against the hashes in the database.

3.4 Searching for Tokens in Network Traffic

By correlating the number of tokens found on the network to the number of tokens redeemed by each team, we were able to get a sense of how much effort teams put into hiding the presence of stolen tokens on the network. This in turn gave us a proxy for how well teams obfuscated traffic to the hosts they have successfully compromised. A commonly used tactic in CTFs is to look for stolen tokens in one's own network traffic as a means of discovering that a token has been stolen. However, for DEF CON 22 CTF, tokens were random strings of numbers and characters of 26 bytes in length, and using a simple pattern-matching approach to identify stolen tokens increases the likelihood of false positives.

In order to reliably and efficiently spot tokens in network traffic, we used two pieces of information obtained from game state data. The first piece of information is the time window in which the token was on the wire. This allowed us to narrow down the number of pcap files in which to look for the token. The second piece of information was the partial token. As explained previously, the game state database only stored alternate characters of the token. Hence, to search for a given token within the network traffic, one would have to use a regular expression to find strings with alternate characters equal to those in the game state database. For example, if the token in the game state database was listed as "zKizlsYvZjxSX," then the regular expression used would be "z.K.i.z.l.s.Y.v.Z.j.x.S.X."

To accomplish the task of finding all the tokens in the network traffic, we developed the following tools:

- SQL string to extract the partial tokens that were redeemed and the time window within which they were redeemed.
- Python script called "insert_dot.py" to create the regular expression from the partial token.
- Bash shell script called "search_flags_in.sh" to search for all redeemed tokens from within the network traffic.
- Bash shell script called "getroundcaps.sh," which find all of the pcap files that contained traffic that corresponded to a given time window.

Figure 3.4 illustrates the process used to search for all redeemed tokens from the DEF CON 22 CTF network traffic.

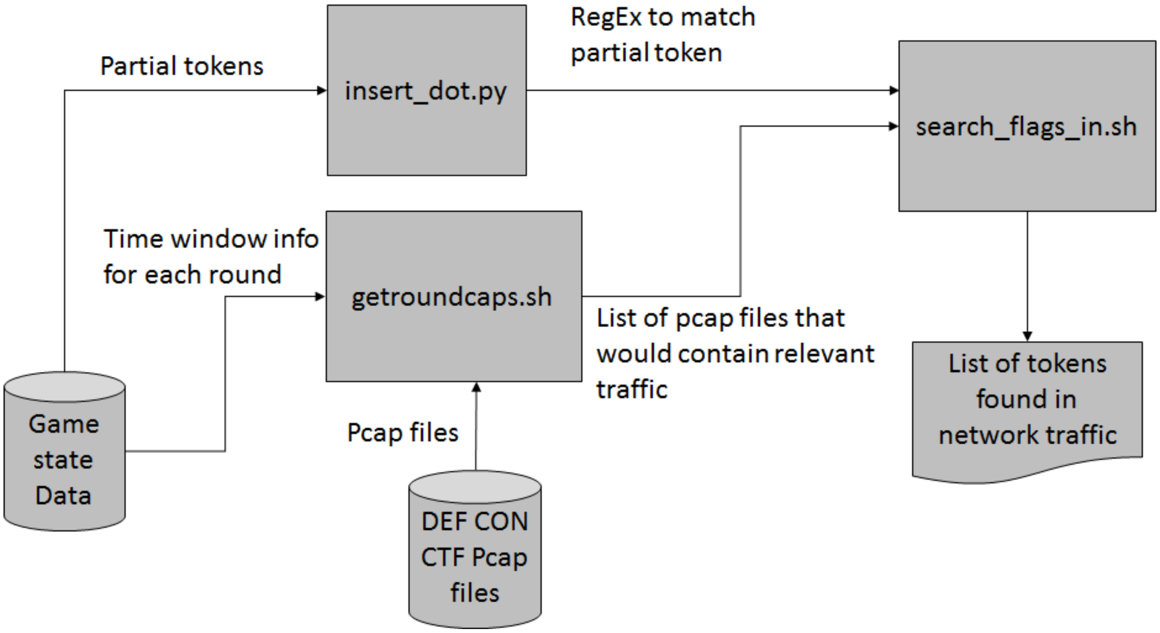

Partial tokens
insert_dot.py
RegEx to match partial token
search_flags_in.sh
Time window info for each round
getroundcaps.sh
List of pcap files that would contain relevant traffic
List of tokens found in network traffic
Game state Data
Pcap files
DEF CON CTF Pcap files

Figure 3.4: Process of finding tokens in DEF CON 22 CTF network traffic

3.5 Searching for Base64 Encoded Tokens in Network Traffic

As part of our research, we wanted to find tokens in network traffic that were Base64 encoded. However, as explained in section 3.4, the complete token was not stored in the database, and therefore we were unable to simply search the network traffic for encoded tokens. Fortunately, we found a a way to spot encoded tokens. All token strings were 26 characters in length, and encoding these tokens resulted in a Base64 encoded string of 36 characters that ended with the letter "K" (due to the padding that was added by the linux Base64 program). There would still be a lot of false positives if we simply accepted all strings of 36 characters that ended with the letter "K." However, we reduced the number of false positives by Base64-decoding the identified string to see if it yielded a valid ASCII string and then subsequently validated the decoded string against the hash of the token stored in the database.

3.6 Searching for Exploits in Network Traffic

Searching for exploits in network traffic was a challenge since we did not have an instance of an attack to use as reference for signature creation. Furthermore, competent attackers, knowing that opposing teams were monitoring network traffic for their exploits, may have obfuscated their exploits to make analysis harder and/or may have sent decoy traffic to make identifying attacks more time consuming.

In order to reliably and efficiently find exploits in the network traffic, we made use of the list of tokens that we had already found within the network traffic. As a by-product of the process of finding the tokens (described in the previous section), we were also able to locate the network flow (which described the flow of data from the server to attacker) that contained the stolen token. Given this, we were also able to find the preceding flow (i.e., the flow from the attacker to the server) and we were able to discover the exploits in one or more of those flows. For example, if we found a stolen token in a flow from 10.5.1.2:8888 to 10.5.2.101:58358, it would imply that host 10.5.2.101 was the attacker and very likely had, at some point in the recent past, sent an exploit over the network. As such, we would need to find the flow that corresponded to host 10.5.2.101:58358 and destination 10.5.1.2:8888 and we would frequently see the exploit in that particular flow. There are some cases where this methodology did not work, such as cases where tokens are sent via a backdoor or cases where the payload of an exploit returns the token over a new connection. In these cases, it was difficult to assert which preceding session was responsible for the actual exploit.

To accomplish the task of finding exploits that returned tokens on established sessions (i.e., not backdoors or exploits that establish a new connection via callback), we developed the following tools:

- A bash script called “extract_exploit.sh” that, when given a network flow that contains a token, found the corresponding network flow containing the exploit that was used to compromise the service and steal the token.

Figure 3.5 illustrates the process used to search for exploits within the DEF CON 22 CTF traffic.

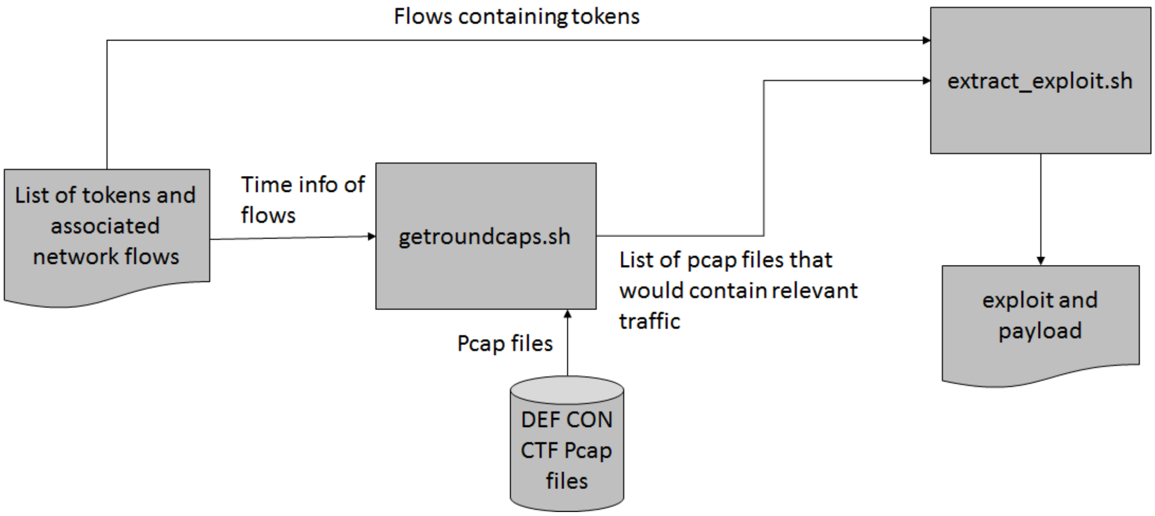


Figure 3.5: Process of extracting exploits from DEF CON 22 CTF network traffic

3.7 Adopting Heuristics to Reduce Work of Studying Exploit Instances

As part of this thesis, we attempted to study two areas related to attack strategies:

1. Investigate if teams in the CTF turn around attacks used against them and reuse them to attack their opponents (commonly referred to as "replay attacks"). In order to do this, we first determined if two exploits are similar. Only then could we determine if an exploit was been taken from one team and reused by another.
2. Investigate if teams used publicly available payloads.

In order to investigate the above two areas, we first extracted and studied each team's exploits. However, throughout the duration of the CTF event, each team launched multiple instances of their exploit against a service. Each instance of the exploit may have been exactly the same or they may have slightly varied to accommodate for changes in the address space of the service or to accommodate for the nature of the vulnerability requiring different input values. As long as the service remained vulnerable to the exploit, the exploit continued to work and teams continued to use the same (or similar) exploit to steal tokens. By the end the CTF, each team may have launched hundreds of instances of each exploit, so searching for and studying every instance of every exploit was not feasible.

While each team may have launched multiple instances of each exploit (each with slight variations), each instance was likely to be almost the same. We used this fact to reduce the number of exploits we needed to study. Hence, instead of studying every instance of an exploit, we assumed that all exploit instances were the same and studied only the first instance of an exploit (i.e., the first exploit that was launched that successfully stole a token). There is a caveat here, and that is services may have multiple vulnerabilities. Therefore, each team may have had one exploit for each vulnerability within the same service, resulting in each team having more than one exploit to attack a single service. By studying only the first successful exploit, we overlooked the other exploits that attacked different vulnerabilities within the same service.

3.8 Similarity Measures for Exploit Data

As explained in the shellcoders handbook [29], an exploit can have two parts—the sequence of bytes that trigger the vulnerability, and the payload that is executed when the vulnerability is triggered and control is handed over to the payload. Determining similarity between two instances of an exploit is different from determining the differences in their payloads alone. As far as we are aware, research done in this area has looked for similarities between exploit code (i.e., payload) rather than the exploit itself. For example, work by done Deguang Kong et al. [30] on exploit code attribution used a statistical model derived from a Markov model for attribution of shellcode. In a similar vein, work done by Manoj Cherukur [31] disassembled shellcodes and computed the mean of Cosine Similarity, Extended Jaccard Similarity and Pearson Correlation measures based on the frequencies of the opcodes to determine similarities of shellcodes. Both pieces of work mentioned above focused on similarities between shellcodes and not the entire exploit. This did not work in our case, where we wanted to compare the entire exploit. In addition, some of the exploits were return-oriented programming (ROP) based, which means that the payload of the exploit did not contain opcodes, but data. Hence, Deguang Kong et al and Manoj Cherukur's work could not be applied to our study, since their work was based on using opcodes to determine similarities.

Determining the similarity of exploits cannot be done by a simple byte-by-byte comparison since the sequence of bytes of an exploit can be somewhat changed without altering its behaviour. An example of this is inserting extraneous new line characters in a text-based exploit. The sequence of bytes has changed from the original exploit, but the exploit may still function as the original. Therefore, a better approach to determining the similarity of exploits was required for this study.

Initially, we used edit distance to measure similarity between exploits. While we found that edit distance was able to indicate if two exploits were dissimilar, it did not tolerate extraneous bytes sequences in similar exploits that were being compared. For example, some exploits were text based and differed because one variant of the exploit had newline characters, whereas the other variant did not. Realistically, we wanted to consider these two exploits as the same, since in these text-based exploits, the newline introduction of newlines characters did nothing to change the nature of the exploit. However, the edit distance measurement was affected by these newline characters. Hence, we did not adopt edit distance as a similarity measure.

In observing the characteristics of an exploit, we found them similar in characteristics of time series data in that:

1. It can tolerate stretching and shrinking of its length and still retain its prominent features (as explained above, in the case of exploits, adding extraneous characters is sometimes possible without affecting the functionality of the exploit).
2. The sequence of the data matters and can be an identifying trait. While exploits do not necessarily require the spatial relationship between byte sequences to be fixed, they do require that the order of byte sequences relative to one another be consistent. For example, some payloads may have the instruction sequence to "push /bin/sh" onto the stack, followed sometime later by an "int 0x80." While the absolute offset from the "push /bin/sh" instruction to the "int 0x80" need not be fixed (i.e., an arbitrary number of instructions can lie between them), the order in which they occur needs to be preserved (i.e., "push /bin/sh" needs to come before "int 0x80").

Given this similarity between time series data and exploit bytecodes, we felt that Dynamic Time Warping (DTW), an algorithm used to compare time series data, would also be suitable to compare exploits. However, we could not simply use the DTW score directly, since we expected to see greater difference for larger exploits. Hence, in order to the account for larger exploits having larger tolerable variances, we divided the DTW by the median size of all the exploits of a given service.

For example, given three exploits for eliza, each developed by *ppp*, *hitcon* and *routards* respectively, we did a pairwise comparison of each of the exploits (i.e., $\binom{N}{2}$). For each comparison, we calculated the DTW distance between the exploits and divided the value by the median size of the three exploits. We considered two exploits to be similar if their DTW ratio is smaller than some threshold value. In order to determine this threshold value, we generated a scatter plot of all the DTW distances for a given exploit and observed that there were clusters. The threshold value was taken as the smallest value that separated one cluster from the other. As an example, look at Figure 3.6. It shows a scatter plot of the normalized DTW distance for the eliza exploit. The x-axis shows the two teams whose exploits are being compared and the y-axis shows the normalised DTW distance. We can see a continuous cluster of dots at the bottom of the scatter plot that are clustered around the normalized DTW distance between 0 and 0.02. Hence, in this case, we would determine the threshold to be 0.02. In other words, if the normalized DTW distance between two exploits is less than or equal to 0.02, we consider these two exploits to be similar.

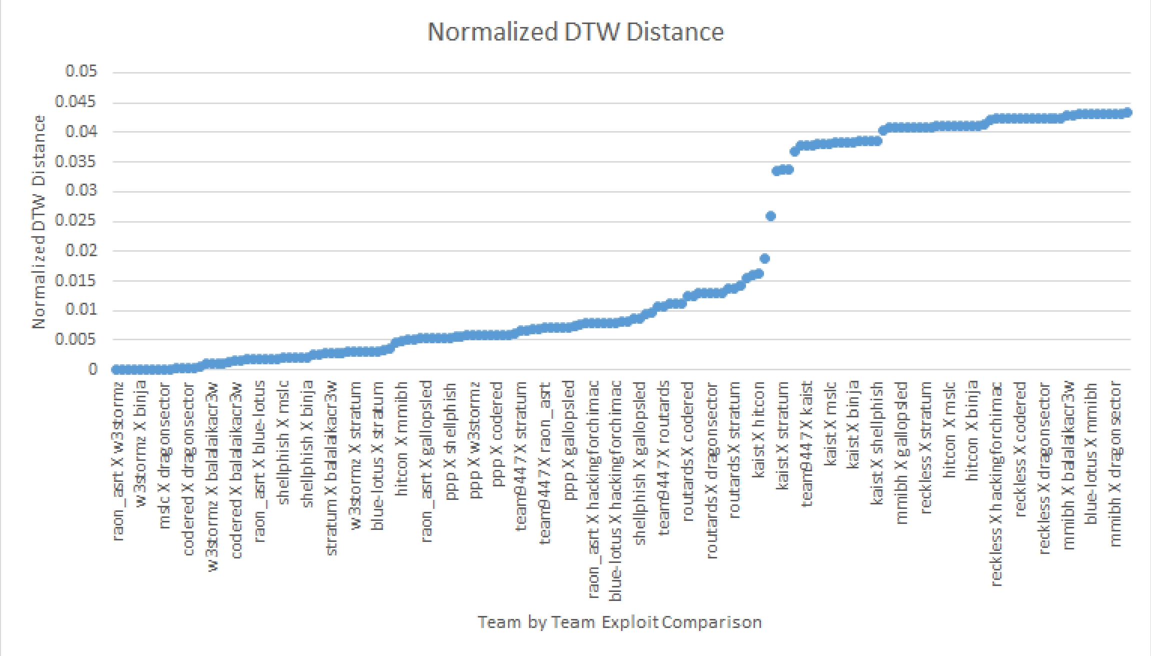


Figure 3.6: Scatter plot of normalized DTW distance of the eliza exploits

3.9 Investigating Use of Exploit Polymorphism

In addition to comparing the exploits of different teams, we also looked at the exploits of a single team throughout the duration of the event. We did so to find out whether teams varied their exploits (perhaps in response to the patching of services) as the game progressed, and if they did, how many exploit variants they created. In order to study whether teams varied their exploits, we looked through every pcap file for each team and extracted every exploit that was successful in stealing a token. We then grouped all the exploits from one team that targeted the same service into one group. Within each group, we took the first exploit as the baseline and compared it to every other exploit. The comparison function we used was DTW. We did this for every team and generated a scatter plot for each team and each service. Figure 3.7 shows an example of the scatter plot for *ppp's* eliza exploit. We can see from the scatter plot that over the course of the CTF, *ppp* used roughly three variants of the exploit, as is visible from the three linear group of dots.

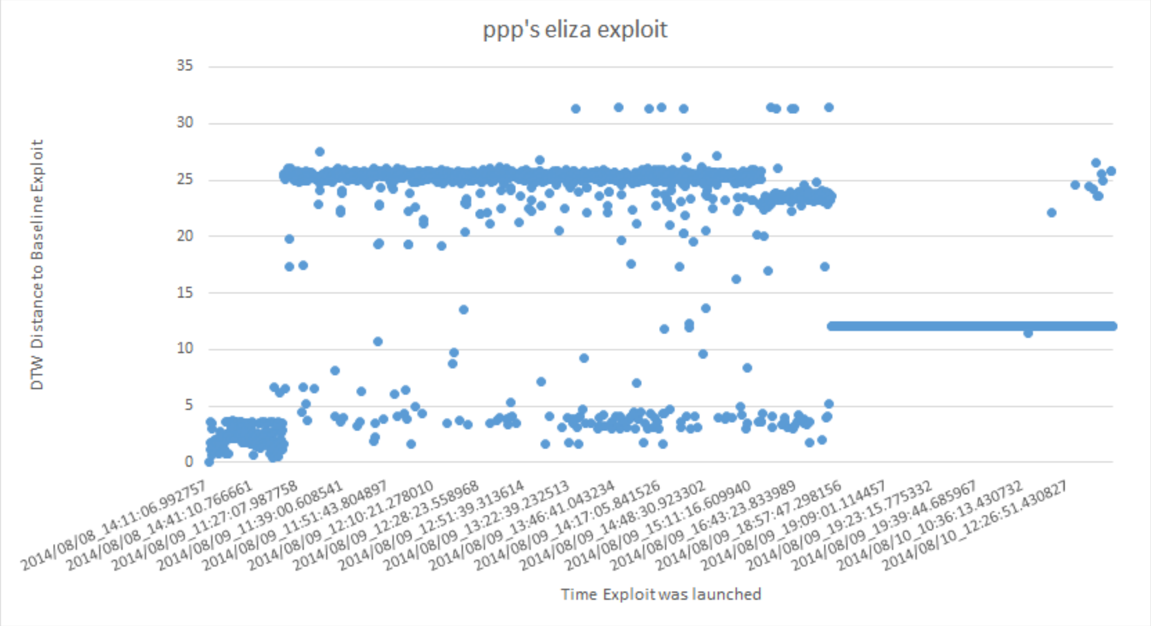


Figure 3.7: Scatter plot of ppp's eliza exploit over the course of the CTF

3.10 Detection of Publicly Available Payloads

One of the metrics we wanted to examine with respect to the competency of teams was their reliance on publicly available payloads. We hypothesized that competent teams developed their own payloads, and it would be a surprising discovery if any of the teams did indeed use publicly available payloads. Before we could determine if teams used publicly available payloads, we first needed to identify and extract the payload from the rest of a given exploit. This was not a trivial task, and in the absence of a reliable and automated way to extract the payload from the exploit, we had to perform the task manually. But once we had all the payloads, we performed a byte-by-byte comparison to determine similarity. Since we were comparing payloads, the methods explained in [30] and [31] could also have been used.

There are many sources of publicly available payloads (Metasploit, for example, is an exploitation toolkit that has a collection of customizable payloads) and the public payloads that we used as references are:

- execve("/bin/sh", ["/bin/sh"], NULL) 30 bytes x64 shellcode from shellstorm [32]
- execve("/bin/sh", ["/bin/sh"], 0) 30 bytes Linux/ARM shellcode from shellstorm [33]
- execve("/bin/sh") Linux/Armle shellcode from Metasploit v4.9.3
- execve("/bin/sh") x86_64 shellcode from Metasploit v4.9.3

If teams used payloads that were similar byte-by-byte to any of the four payloads mentioned above, we determined that the team has used a publicly available payload.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 4: RESULTS AND ANALYSIS

This chapter covers findings from our study on the network traffic capture files downloaded from [34]. The chapter consists of three parts. The first part describes interesting discoveries that we made through data visualization and manual packet inspections. The second part describes the metrics we developed in an attempt to correlate team strategies to their final game standing. The third part of the chapter provides an analysis of the metrics.

4.1 Interesting Discoveries

This section describes the interesting discoveries resulting from our analysis. The methodology used to make these discoveries involved visualizing the network traffic data to spot interesting or anomalous artifacts along with a manual analysis of the captured network packets.

4.1.1 Use of Automation for Attacks

Figure 4.1 shows the number of bytes sent per second to the vulnerable services for Round 91 and Round 92. The red line indicates where Round 91 ended and Round 92 began. We chose Round 91 and 92 as an example to highlight. These two rounds were near the end of Day One, and we expected teams to have settled into their routines by then. We also looked at the traffic patterns of subsequent rounds in Day Two and Day Three of the competition, and we also saw similar patterns.

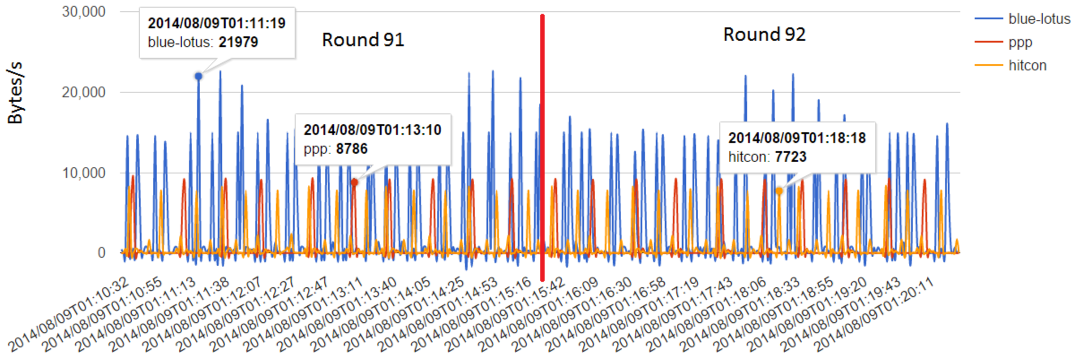


Figure 4.1: Line chart showing bytes sent by three of the top five teams

Looking at the chart we observe that:

- Three of the top five teams had regular spikes in their network traffic. Figure 4.2 shows only traffic for Round 91 and Round 92 that originated from *hitcon's* clients and was destined for the ports of vulnerable services of which other teams were listening. We can see that there was periodic traffic sent to the elsa and wdub service. Similarly, in *ppp's* and *blue-lotus'* traffic, we see that they too had regular spurts. Figure 4.3 shows only network traffic that originated from *ppp* that were destined to the vulnerable services. We can clearly see that there were regular spikes of traffic going to the wdub service. Likewise, Figure 4.4 shows only network traffic that originated from *blue-lotus* that was destined for the vulnerable services. We can see that there are regular spikes of traffic going to the eliza and wdub service.

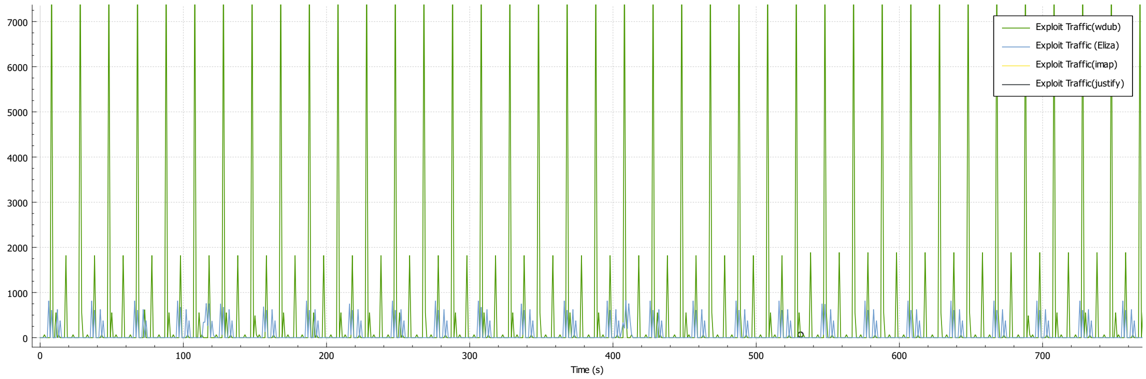


Figure 4.2: Line chart showing bytes of exploit traffic sent by hitcon

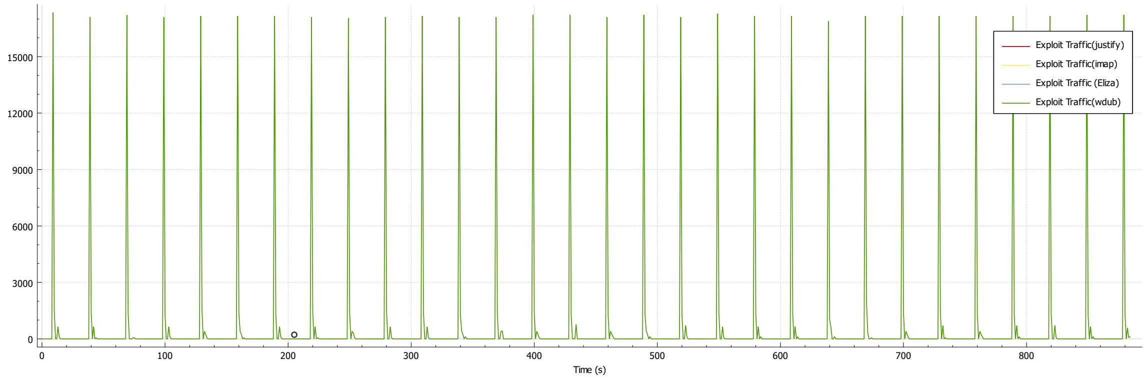


Figure 4.3: Line chart showing bytes of exploit traffic sent by ppp

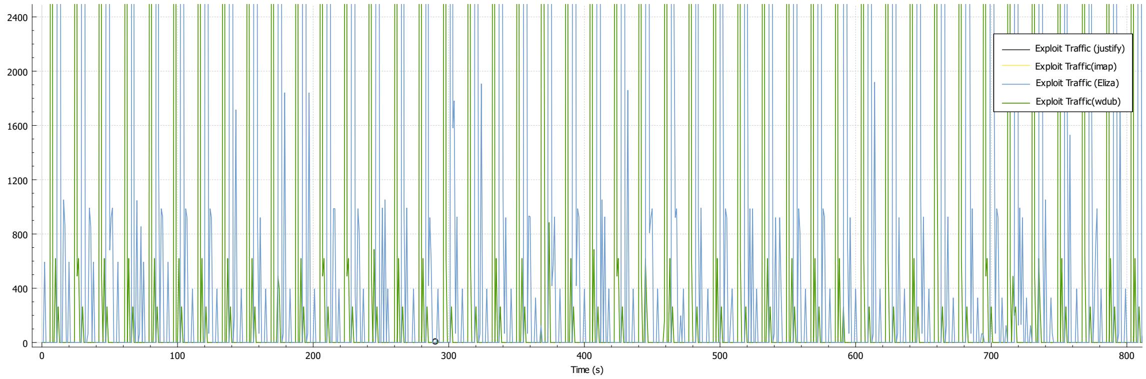


Figure 4.4: Line chart showing bytes of exploit traffic sent by blue-lotus

- From Figure 4.1, there appeared to be no difference in the frequency of launching exploiting between one round and the next. For three of the top five teams, we did not see any special activities taken by the teams as they transit from Round 91 to Round 92. However, this does not mean that in subsequent rounds the teams did not modify their behavior. One area of future work to look at would be to study how teams change their behavior as they progress through the CTF event.
- Comparing Figure 4.2, Figure 4.3 and Figure 4.4, we see that the frequencies that they launched their respective exploits were different. *blue-lotus* appears to have had a higher frequency of launching exploits compared to *hitcon* and *ppp*. However, this does not mean that the teams did not vary their behaviors as they progress through the CTF event. In Section 4.2, we looked at the rate of exploitation to study the frequency at which teams launched their exploits.

From our observations above, it seems likely that teams used automation to launch their exploits at regular intervals. This makes sense since each team needed to launch their exploits at regular intervals (based on round lengths) during the competition and automating this process released manpower for other tasks.

4.1.2 Scanning and Exploiting Hosts

We examined the number of connections initiated by clients of one to clients of other teams (i.e., only client-to-client connections) to discover if teams attempted to connect to and/or exploit the clients of other teams. We discovered four hosts that initiated a large number of connections to other clients, which we deemed suspicious, since we expected clients to only attack servers running vulnerable services. The four hosts (belonging to three teams) were 10.5.9.102, 10.5.9.106, 10.5.14.107 and 10.5.18.21. The first two hosts belonged to *hitcon*, the 3rd belonged to *mslc* and the last host belonged to *gallopseld*. Table 4.1 shows the number of connections initiated by each host.

Table 4.1: Number of client-to-client connections initiated by hitcon, msls and gallopsled

Team	Client	Number of Outgoing Connections
hitcon	10.5.9.102	4869
	10.5.9.106	4867
mslc	10.5.14.107	4445
gallopsled	10.5.18.21	3843

Since the number of outgoing connections for the four hosts seemed anomalous, we decided to investigate further and learned that:

- The various hosts (i.e., 10.5.9.102, 10.5.9.106, 10.5.14.107 and 10.5.18.21) were conducting port scans against various clients.
- *Hitcon* made several attempts to exploit 10.5.18.114. Manually inspecting *hitcon's* packet captures, we learned that *hitcon* discovered a phpMyAdmin site running on 10.5.18.114 and was attempting to exploit one of the clients. *Hitcon* tried several urls such as "`pma/main.php`," "`pma/login.php`," "`pma/index.php`" and even tried an SQL injection attack with the string "`/PMA/main.php?reload=1&sql_query=select+111+into+outfile+%27C%3A%5C%5CAPM_Setup%5C%5CServer%5C%5Capache%5Chtdocs%5Cas.txt%27&token=7bdf3857c49f1da281915b7105fcc00f.`" However, we do not see any traffic indicating that *hitcon* was able to execute any commands on the remote host, and we concluded that *hitcon* was unsuccessful in exploiting the host 10.5.18.114.

From our observations above, it seems that teams did not limit their attacks to the organizer-supplied servers. Any computer connected to the network was fair game.

4.1.3 Suspicious Traffic Between Teams

We created network connection graphs that showed client-to-client connections to visually aid us in spotting anomalous activities. From the network graph consisting of client-to-client connections that involved *hackingforchimac* clients, shown in Figure 4.5, we discovered that host 10.5.14.107 made several connections to clients belonging to *mslc*. The color of the nodes represent the teams that owned the nodes (red nodes represent *hackingforchimac* and blue nodes represent *mslc*). The direct edges represent flows from one host to another (all flows with the same origin and destination are represented by one edge). At the bottom left, we see a red node representing host 10.5.14.107 initiating connections to a number of blue nodes representing hosts belonging to *mslc*.

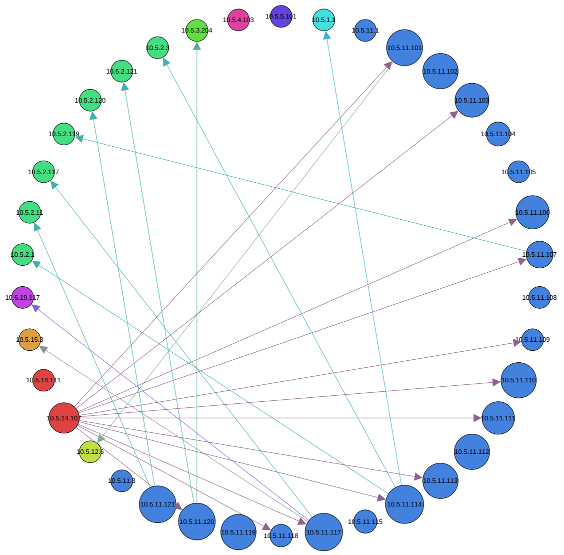


Figure 4.5: Network graph of hosts communications involving hackingforchimac

Manually inspecting the network traffic associated with these connections, we discovered a large number of packets sent between host 10.5.11.113 (belonging to *hackingforchimac*) and 10.5.14.107 (belonging to *mslc*). A partial dump of one of the packets is shown below:

```
00000000  16 03 01 02 6f 02 00 00  46 03 01 53 e6 9f 68 fb  |....o...F..S..h.|
00000010  52 08 e4 9a 90 7f 00 c1  b9 5f 34 fd 5c 4f d5 15  |R........_4.\O..|
00000020  f9 2b 82 af 9d 57 71 a2  34 8a 3b 20 af 15 00 00  |.+...Wq.4.; ....|
00000030  de ed 80 b3 ad 73 c5 c7  62 78 fa d0 73 f9 2d fb  |.....s..bx..s.-.|
00000040  f7 a6 8e d7 91 25 ff 7d  9a 1f 47 df 00 2f 00 0b  |.....%.}..G../..|
00000050  00 02 1d 00 02 1a 00 02  17 30 82 02 13 30 82 01  |.........0...0..|
00000060  7c a0 03 02 01 02 02 10  35 11 c4 f9 00 f1 9e 80  ||.......5.......|
00000070  46 81 27 39 45 43 49 21  30 0d 06 09 2a 86 48 86  |F.’9ECI!0...*.H.|
00000080  f7 0d 01 01 05 05 00 30  48 31 46 30 44 06 03 55  |.......0H1F0D..U|
00000090  04 06 13 3d 55 53 2c 53  54 3d 43 41 2c 4c 3d 53  |...=US,ST=CA,L=S|
000000a0  61 6e 20 46 72 61 6e 63  69 73 63 6f 2c 4f 3d 42  |an Francisco,O=B|
000000b0  69 74 54 6f 72 72 65 6e  74 2c 4f 55 3d 75 54 6f  |itTorrent,OU=uTo|
000000c0  72 72 65 6e 74 2c 43 4e  3d 75 54 6f 72 72 65 6e  |rrent,CN=uTorren|
000000d0  74 30 1e 17 0d 31 34 30  37 30 38 31 33 33 39 35  |t0...14070813395|
000000e0  32 5a 17 0d 31 35 30 37  30 38 31 39 33 39 35 32  |2Z..150708193952|
000000f0  5a 30 48 31 46 30 44 06  03 55 04 06 13 3d 55 53  |Z0H1F0D..U...=US|
00000100  2c 53 54 3d 43 41 2c 4c  3d 53 61 6e 20 46 72 61  |,ST=CA,L=San Fra|
00000110  6e 63 69 73 63 6f 2c 4f  3d 42 69 74 54 6f 72 72  |ncisco,O=BitTorr|
00000120  65 6e 74 2c 4f 55 3d 75  54 6f 72 72 65 6e 74 2c  |ent,OU=uTorrent,|
00000130  43 4e 3d 75 54 6f 72 72  65 6e 74 30 81 9f 30 0d  |CN=uTorrent0..0.|
00000140  06 09 2a 86 48 86 f7 0d  01 01 01 05 00 03 81 8d  |..*.H...........|
00000150  00 30 81 89 02 81 81 00  f4 bd 8e a2 56 d0 7e 8c  |.0..........V.~.|
00000160  33 38 51 e3 a3 9f 27 6a  4d df 92 c8 d9 b5 31 9d  |38Q...’jM.....1.|
00000170  14 b4 d1 de b4 74 7b f1  d1 0f 80 f3 8d bc 48 26  |.....t{.......H&|
00000180  92 68 cf 56 da 53 6f 7d  65 82 6a f3 b4 fc 4c bb  |.h.V.So}e.j...L.|
00000190  d3 c7 89 35 67 a6 eb 94  9a a6 29 79 5b 17 2e 53  |...5g.....)y[..S|
000001a0  08 4e 6b cc 36 7e f0 16  38 48 55 01 f6 54 be 02  |.Nk.6~..8HU..T..|
000001b0  01 f6 2c 09 c0 bf 37 e4  89 f5 13 56 13 d0 aa e7  |..,...7....V....|
000001c0  dc fd 8a b1 a2 e0 38 f2  9c 68 1f 62 67 c8 5a 02  |......8..h.bg.Z.|
000001d0  12 73 b4 2d 51 e5 c7 71  02 03 01 00 01 30 0d 06  |.s.-Q..q.....0..|
000001e0  09 2a 86 48 86 f7 0d 01  01 05 05 00 03 81 81 00  |.*.H............|
```

The contents of the packets seemed to indicate that the traffic is secure Bittorrent negotiation traffic as it contained what looks like a x.509 certification with *uTorrent* as a canonical name. We are unable to explain why two clients of opposite teams would be communicating in such a fashion. There is a possibility that one of *hackingforchimac’s* clients was compromised and the traffic we see is obfuscated data that was being ex filtrated. It is also possible that these two teams were collaborating, and the traffic we see is them exchanging information. We leave this as area for future research.

4.1.4 Trivial Backdoors

We wanted to study whether teams installed backdoors in the servers that they exploited. Hence, we wrote a script to look for callbacks in the network traffic (i.e., traffic initiated by the server to send tokens back to clients that had exploited it). Our script discovered evidence of trivial backdoors. These "backdoors" are simple shell scripts that ran periodically. The shell script below is an example of the callback that *raon_asrt* used to periodically send tokens.

```
while true; do nc 10.5.5.101 22212 < /home/eliza/flag; sleep 60; done.
```

4.2 Metrics

This section describes the metrics we used in our attempt to correlate teams strategies to their final standing in the event. Though some of these metrics may indicate correlation with the team's final standing, they by no means indicate causation. These metrics are derived from the network traffic captures and the game state data from the scoring servers. Table 4.2 shows the final standing of each team and their respective scores (which were determined by the flags they owned at the end of the CTF competition).

Table 4.2: Table showing the final standing and score of each team

Team	Rank	Score
ppp	1	11263
hitcon	2	7833
dragonsector	3	4421
reckless	4	4020
blue-lotus	5	3233
mmibh	6	2594
raon_asrt	7	2281
stratum	8	1529
team9447	9	1519
kaist	10	1334
routards	11	1262
mslc	12	1248
binja	13	1153
codered	14	997
w3stormz	15	987
penthackon	16	979
balalaikacr3w	17	937
gallopsled	18	921
shellphish	19	899
hackingforchimac	20	546

Rank and Score Information taken from [35]

4.2.1 Teams' Offensive Capabilities

In this subsection, we investigate how much a team's offensive capabilities contributed to its success in the CTF competition. The metrics we used to determine offensive capabilities are:

- Number of tokens redeemed

- Percentage of tokens found on wire during ex-filtration
- Average time to develop an exploit
- Use of publicly available payloads
- Exfiltration techniques
- Rate of exploitation
- Variation of attack parameters
- Exploit polymorphism

Number of Tokens Redeemed

One of the most straightforward metrics to measure the offensive capabilities of a team was the number of tokens that each team had redeemed. Intuitively, a large number of tokens redeemed implies that the team was able to find vulnerabilities and successfully exploit them. However, it is important to note that the number of tokens redeemed did not necessarily equate to winning more flags. This was due to two reasons:

- **Flags were shared between any teams that successfully exploited a service and redeemed a token.** As explained in Section 2.5, for a given round, flags were equally distributed among teams that stole and redeemed a token from a service they exploited. Hence, if many teams redeemed a token for a given round, then each of these teams would have had a smaller share of the flags since the flags are distributed equally among them. However, if a team was the only team that redeemed a token for a given round, then all the flags associated with the service were given to the team.
- **Teams could lose flags.** As explained in Section 2.5, teams that had their services exploited for a given round lost a portion of the flags associated with the service. Teams can also lose flags for not keeping their services up.

Table 4.3 shows the number of tokens redeemed by each team.

Table 4.3: Table showing the number of tokens redeemed by each team

Team	Rank	No. of Tokens Redeemed
ppp	1	2411
raon_asrt	7	1589
hitcon	2	1531
reckless	4	1447
mslc	12	1155
blue-lotus	5	1136
mmibh	6	811
kaist	10	803
dragonsector	3	778
codered	14	742
routards	11	722
team9447	9	662
stratum	8	615
gallopsled	18	587
hackingforchimac	20	576
balalaikacr3w	17	537
penthackon	16	489
w3stormz	15	478
shellphish	19	386
binja	13	234

Percentage of Tokens Found on Wire During Ex-filtration

One strategy that teams employ is to capture and re-use exploits off the wire. However, identifying exploits on the wire can be difficult. Hence, most teams adopt the approach of monitoring the network for stolen tokens leaving their servers. Tokens leaving their server indicate that a service has been successfully compromised, and finding the corresponding exploit becomes a matter of examining network data sent prior to the point the tokens were leaked. In the case of DEF CON 22 CTF, tokens were random strings; hence, there was no one signature that a team could use to detect all tokens leaving their servers. They could however, write scripts that read their own tokens every round to find out what their token was and then look for those tokens in their egress traffic to detect leaked tokens.

Given that teams were actively looking for exploits and tokens on the wire, it makes sense that competent teams obfuscated their ex-filtration traffic, so that the tokens that they stolen were not visible on the wire. It also stands to reason that preventing other teams from stealing and re-using one's exploit was a good way to maintain one's standing in the competition. Hence, we examined the number of un-obfuscated tokens stolen by a team to identify any correlation between this strategy and a team's final ranking. For the purposes of this metric, we also considered tokens that were trivially encoded using the Base64 encoding scheme to be un-obfuscated.

Table 4.4 shows the number of tokens redeemed by each team versus the number of tokens found on the network (either in the clear or trivially encoded using the Base64 encoding scheme). Any tokens unaccounted were counted as obfuscated using stronger methods or possibly obtained via out-of-band means (such as trading with other teams). As future research, we could study the exploit payloads in more detail to determine what type of obfuscation techniques teams employ.

Table 4.4: Table showing the number of tokens seen on the wire (in clear or Base64 encoded) versus the number of tokens redeemed

Team	Rank	No. Of Tokens Redeemed	No. Of Tokens Found (In Clear)	No. Of Tokens Found (Base64)	No. Of Tokens Not Accounted	Percentage Of Tokens Found
hitcon	2	1531	760	23	748	51.14%
reckless	4	1447	805	23	619	57.22%
gallopsled	18	587	295	80	212	63.88%
routards	11	722	566	0	156	78.39%
binja	13	234	170	31	33	85.90%
kaist	10	803	772	0	31	96.14%
mslc	12	1155	1118	0	37	96.80%
codered	14	742	728	0	14	98.11%
balalaikacr3w	17	537	532	0	5	99.07%
ppp	1	2411	2218	173	20	99.17%
blue-lotus	5	1136	1018	113	5	99.56%
w3stormz	15	478	476	0	2	99.58%
dragonsector	3	778	775	0	3	99.61%
raon_asrt	7	1589	1583	0	6	99.62%
mmibh	6	811	809	0	2	99.75%
hacking-forchimac	20	576	575	0	1	99.83%
stratum	8	615	615	0	0	100.00%
team9447	9	662	631	31	0	100.00%
penthackon	16	489	489	0	0	100.00%
shellphish	19	386	283	103	0	100.00%

It should be noted that there are teams that obfuscated some tokens but not others. Table 4.5 shows for each service, how many of the stolen tokens were Base64 encoded. We can see that teams like *ppp* and *9447* only Base64-encode tokens stolen from one type of exploits and teams like *blue-lotus* and *gallopsled* Base64-encoded tokens stolen from two service.

There were no teams that Base64-encoded tokens stolen from all services. One area of future work would be to investigate why teams only obfuscate the stolen tokens for some services and not others (could it be that for some exploits, obfuscation is not possible?) and what factors contribute to a team's decision to obfuscate.

Table 4.5: Table showing the breakdown by service of the tokens that were Base64 encoded

	No. Of Tokens Obfuscated			
TeamName	eliza	wdub	justify	imap
ppp	0	0	0	173
team9447	0	0	31	0
reckless	11	0	12	0
routards	0	0	0	0
raon_asrt	0	0	0	0
kaist	0	0	0	0
shellphish	103	0	0	0
codered	0	0	0	0
hitcon	23	0	0	0
blue-lotus	0	0	84	29
hackingforchimac	0	0	0	0
mmibh	0	0	0	0
w3stormz	0	0	0	0
mslc	0	0	0	0
dragonsector	0	0	0	0
penthackon	0	0	0	0
stratum	0	0	0	0
gallopsled	58	0	22	0
balalaikacr3w	0	0	0	0
binja	0	0	31	0

Average Time to Develop an Exploit

How quickly a team is able to develop a successful exploit may indicate how skilled a team is at vulnerability discovery and exploit development. The measurements for this metric were derived from game state data and not from the network packet captures. We used the round in which a team first successfully redeemed a token as a proxy for when it first successfully exploited a service. The assumption here was that a team launched an exploit as soon as they had a working exploit (i.e., they do not delay using an exploit). If a team was not able to exploit a given service throughout the event, then we used the value of 273 as the Average Time to represent this fact (there are 272 rounds in total for the entire CTF, hence we chose a value of 273 to indicate that a particular team failed to exploit the service).

Table 4.6: Table showing exploit development time (in terms of Rounds) for each team

		Exploit Development Time (in Rounds)					
TeamName	Rank	eliza	wdub	badger	justify	imap	T_{avgED}
hitcon	2	32	54	273	162	93	122.8
ppp	1	51	60	273	155	113	130.4
raon_asrt	7	36	70	273	166	110	131
blue-lotus	5	43	69	273	171	113	133.8
stratum	8	82	63	273	206	113	147.4
codered	14	44	63	273	219	163	152.4
team9447	9	68	63	273	246	117	153.4
reckless	4	139	63	273	205	114	158.8
dragonsector	3	58	87	273	273	119	162
mslc	12	46	63	273	161	273	163.2
kaist	10	118	70	273	219	137	163.4
routards	11	118	77	272	243	130	168
balalaikacr3w	17	100	63	273	174	273	176.6
shellphish	19	84	238	273	226	113	186.8
w3stormz	15	58	63	273	273	273	188
hackingforchimac	20	63	63	273	273	273	189
mmibh	6	66	70	273	266	273	189.6
binja	13	54	244	273	228	273	214.4
penthackon	16	273	66	273	273	195	216
gallopsled	18	118	226	273	225	273	223

Use of Publicly Available Payloads

Using publicly available payloads for their exploits may indicate a lack of sophistication on the part of the team. Intuitively, we expected that sophisticated teams developed custom payloads to achieve effects unique in a CTF environment. As mentioned in Section 3.10, we compared the exploit payloads to the following publicly available payloads:

- execve("/bin/sh", ["/bin/sh"], NULL) 23 bytes x86 shellcode from shellstorm [32]
- execve("/bin/sh","/bin/sh",0) 30 bytes Linux/ARM shellcode from shellstorm [33]
- execve("/bin/sh") Linux/Armle shellcode from Metasploit v4.9.3
- execve("/bin/sh") x86_64 shellcode from Metasploit v4.9.3

We did not detect the use of any of the above mentioned payloads. However, it should be noted that the method used to compared the payloads was an extremely naïve byte-by-byte comparison. This means that if a teams modified the payload even by even a single byte, we would not have detected the payload. Also, we only compared the first four payloads, which is extremely limited. Teams may have used any number of publicly available payloads to include bind shells and reverse shells. We did not search for such payloads as they look for configurable parameters (specifically the reverse connect IP address and port number) which our byte-by-byte comparison method was not equipped to handle. In future work, we could propose better methods for detecting the use of publicly available shellcodes, such as those detailed in [31].

Ex-filtration Techniques

For this metric, we considered only two techniques used by teams to ex-filtrate flags. The first technique involved reusing existing connections to send tokens back to the attacking host, and the second technique created a second connection that called back to the attacker from the exploited server. Callback payloads make it more difficult to associate the network flow that ex-filtrated the token to the flow that contains the exploit. Thus, callback payloads may be a good anti-intrusion detection measure as they required additional time on the part of the defender to sieve out the associated exploit from the network packet captures.

It should be noted that in gathering measurements for this metric, we only studied payloads that either sent tokens in the clear, or sent them as Base64-encoded data. If more sophisticated obfuscation techniques were used, we could not tell if the data on the network was an ex-filtrated token, and we could not ascertain whether or not the payload was a callback.

Payloads that ex-filtrated tokens by reusing existing connections had flows with source port belonging to one of the services (such as 143, 4444, 6969 or 8888). Hence, we identified callbacks by flows that contained the token, had a server source IP address and a non-service source port.

Table 4.7: Table showing the type of payloads used by each team

Team	Session Reuse	Callback
ppp	X	
hitcon	X	
dragonsector	X	X
reckless	X	
blue-lotus	X	X
mmibh	X	
raon_asrt	X	X
stratum	X	
team9447	X	X
kaist	X	X
routards	X	X
mslc	X	X
binja	X	
codered	X	X
w3stormz	X	
penthackon	X	
balalaikacr3w	X	X
gallopsled	X	X
shellphish	X	
hackingforchimac	X	

It is interesting to note that for the top two teams *ppp* and *hitcon*, we did not find them using callbacks. However, as mentioned, it does not mean that these two teams did not use callback payloads, but that we simply failed to find the network traffic that contained the tokens they ex-filtrated using callbacks. This is entirely possible since from Table 4.4, we know that both *ppp* and *hitcon* had tokens that we were unable to find in the network data.

Rate of Exploitation

From observations made in Subsection 4.1.1, we determined that teams were automating their exploit tasks. We were interested to find out if the rate at which teams exploit a service had any bearing on their finals standing. In a given round, a team is allowed to redeem a token at most one time, hence launching an exploit multiple times per round against the same target yields at most one score (as you would only manage to steal the same token again and again). In fact, launching exploits indiscriminately increased the chances of detection and potential reuse.

There are legitimate scenarios where teams make multiple exploit attempts per round. One possible scenario is for redundancy, where teams are unable to ascertain from the scoring server if their exploits were successful. Another scenario is when teams are not aware of when a round begins, and they keep launching their exploits to avoid missing the opportunity to score points in a round.

To calculate the average number of exploits launched per round, we searched through the network packets and counted the number of exploit attempts that successfully returned a token that was subsequently redeemed. We then took this number of exploit attempts and divided it by the number of active rounds. An active round was defined as a round where, according to the game state data, a team has redeemed at least 1 token. Table 4.8 shows the average exploit rate of each team.

Table 4.8: Table showing average exploit rate (in terms of exploits per round) of each team

Team	Total Number of Exploit Instances Launched in Event	Number of Active Rounds	Average Exploit Rate (in exploit instance per round)
ppp	64445	208	309.8317
hitcon	13481	209	64.50239
dragonsector	9904	164	60.39024
reckless	16659	183	91.03279
blue-lotus	48076	207	232.2512
mmibh	8737	150	58.24667
raon_asrt	31721	218	145.5092
stratum	27853	178	156.4775
team9447	10148	141	71.97163
kaist	5444	164	33.19512
routards	6627	146	45.39041
mslc	73016	185	394.6811
binja	2678	45	59.51111
codered	16929	164	103.2256
w3stormz	9908	117	84.68376
penthackon	7758	89	87.16854
balalaikacr3w	80537	149	540.5168
gallopsled	2508	85	29.50588
shellphish	7327	153	47.88889
hackingforchimac	62113	154	403.3312

Since there were twenty teams, each running four services, an ideal rate of exploitation was 76 exploits per round (1 exploit for each service of each team). Comparing the exploitation rate of the top five teams and the bottom five teams, we found that they had exploitation rates that ranged from under 30 exploits per round to over 400 exploits per round.

It should be noted that we see a wide range of exploitation rates. The top team, *ppp* had a high rate of exploitation at about 309.8 exploits per round. This works out to be about 3.8 times more than the ideal rate. On the other hand, *hitcon*, who came in 2nd place, had a seemingly much lower rate of exploitation of only 64.5 exploits per round. However, we could not find a large number of *hitcon's* ex-filtrated tokens (we ony found 54.14% as opposed to 99.17% of *ppp's*) and the rate of exploitation is determined over the exploits that we have found to have successfully stole tokens. Therefore, *hitcon's* rate of exploitation may possibly be higher than what we recorded.

Variation of Attack Parameters

To measure how a team varied its attacks, we examined the following two behaviours:

- The number of hosts it used to attack
- The number of different callback ports used for reverse connect payloads

In order to acquire the number of hosts a team used to attack the other teams, we counted the number of unique destination IP addresses in flows that originated from servers and contain tokens. The assumption here is that if a flow originated from a server and contained a token, then the destination host must have exploited the server or was used as a callback listener. In either case, we considered the host as being involved in the attack on the server.

As for the different number of callback ports used, we derived this by searching for flows that had a source IP address belonging to a server and a source port that is not used by any of the vulnerable services (i.e., a source port that is not 143, 4444, 6969 or 8888). The assumption here is that if a team reuses an existing connection in an attack, the source port for flows going from the server to the attacker would be one of the ports used by the vulnerable services. Hence, if we saw a flow that originated from a server that had a source port that was not one of the service ports, then the server must have been the one to initiate the connection and the destination port must have been a port used by a listener. One point to note is that teams also use their servers to exploit other servers; hence, if we relied solely on looking for servers that initiated connections to determine callback ports used, we would erroneously include exploits attempts. Tables 4.9 and 4.10 show the number of attacking hosts used by each team and the number of different callback ports each team used, respectively.

Table 4.9: Table showing the number of attacking hosts used by each team

Team	Number of Attacking Hosts
ppp	3
hitcon	2
dragonsector	2
reckless	2
blue-lotus	4
mmibh	2
raon_asrt	2
stratum	2
team9447	4
kaist	4
routards	5
mslc	4
binja	3
codered	2
w3stormz	2
penthackon	3
balalaikacr3w	6
gallopsled	4
shellphish	2
hackingforchimac	12

From Table 4.9, we can see most teams used two to four hosts to launch their attacks. This could imply some level co-ordination within the teams (e.g., a distribution of roles or used of designated hosts for attacking), since each team consisted of more than four members and we expected to see more than four attacking hosts if every team member were to work on their own to launch attacks. There were two outliers—*balalaikacr3w* and *hackingforchimac*. These two teams used a large number of hosts for attacking, but it appears that the strategy did not work too well for them, considering that they were ranked within the last four of the CTF event.

Table 4.10: Table showing the number of different callback ports used by each team (Teams that do not use callbacks are not shown)

Team	Number of Callback Ports	Ports Used
dragonsector	2	16637, 18877
blue-lotus	5	7331, 9876, 9877, 9878, 9879
raon_asrt	3	15554, 22212, 22213
team9447	1	10101
kaist	1	44444
routards	7	1337, 1338, 4444, 8888, 8890, 8891, 8892
mslc	3	1010, 10101, 1013
codered	1	1337
balalaikacr3w	3	1489, 1514, 1518
gallopsled	28	1336, 34934, 35206, 37901, 38655, 40212, 41738, 42031, 43431, 45426, 48890, 49515, 51005, 51537, 51870, 51908, 52803, 52910, 53339, 53474, 54154, 54957, 55083, 57489, 57702, 58846, 58952, 59095

From Table 4.10, we see that most teams that used callback payloads generally used callback ports that had some pattern to them. For example, *blue-lotus* used the port 7331 (which is a moniker for "elite" in hacker speak). They also used sequential port numbers 9876, 9877, 9878 and 9879. *raon_asrt* used port numbers that had lots of repeating digits such as 15554, 22212 and 22213. *gallopsled's* method of assigning callback ports appread different from the rest. Their port assignments appeared more random, though a large number of these ports were in the 40 thousand to 50 thousand range. Observing the data from Table 4.10, if the callback ports were not truly randomized, it may have been possible for a team to detect that its tokens were being ex-filtrated by looking at the callback port of the ex-filtration traffic. Hence, a good strategy would be to use random callback ports.

Exploit Polymorphism

As the CTF event progressed, teams could change their exploits to react to changing situations. For example, teams could change their exploits in anticipation that other teams developed signatures for their exploits and block their attacks. In addition, vulnerabilities that teams were originally targeting could have been patched, so they had to develop another exploit to attack a different vulnerability.

For this metric, we looked at how the exploits of each team changed over the course of the CTF event. For each team, we extracted from the packet captures all their successful exploits that targeted the same service. A successful exploit was an exploit that successfully exploited the service it was targeting and stole a token. We then compared the exploits against a baseline exploit for the service. This baseline exploit was the first exploit that we found on the network that had successfully exploited the service.

In Figure 4.6 we see the scatter plot for the wdub exploits of the top two ranked teams *ppp* and *hitcon*. The scatter plot shows the DTW distances of all the exploits targeting wdub compared to the baseline exploit (which is the first successful wdub exploit of each team found on the network). The y-axis of the scatter plot marks the DTW distance while the x-axis marks the time the exploit was launched. On the scatter plot, each linear group of dots represent an exploit variant. From the scatter plots, we can see that as the CTF progressed, there were periods where *ppp* and *hitcon* used different exploits variants, as indicated by the disjointed linear grouping of dots. In addition, from the step like nature of the scatter plot, it seemed that *ppp* and *hitcon* do not use previous variants once they have launched they new variants. We can also see that the teams may have simultaneously used two variants (as noted by the two parallel linear group of dots).

We contrast this to Figure 4.7, where we see the scatter plot for the wdub exploit for the bottom two ranked teams *shellphish* and *hackingforchimac*. For both these teams, we can see parallel linear groups of dots (just like with *ppp* and *hitcon*), indicating they may have used two variants simultaneously. We can observe that *hackingforchimac* appeared to have used only one variant for the most part, except near the end of the CTF where it appeared to have about four variants which it used simultaneously. As for *shellphish*, they only launched their wdub exploits late in the game (on the third day of the competition); hence, we could not draw any conclusions from their scatter plot.

The fact that *ppp* and *hitcon* did not use previous variants once they started using a new variant may indicate that these teams knew something about the state of the game (e.g., that teams may have patched one of the vulnerabilities in their services). This implies that these two teams may have had good situational awareness. In contrast, *hackingforchimac* used the same variant all the way through to near the end of the CTF. This may indicate that a lack of situational awareness and they may have used exploits that did not work against all teams. A quick check against the game state database seemed to support this conclusion. Beyond round 89, there were teams that *hackingforchimac* did not redeem tokens from their wdub service, whereas *ppp* was able to.

Figure 4.6: Variants of ppp's and hitcon's wdub exploit

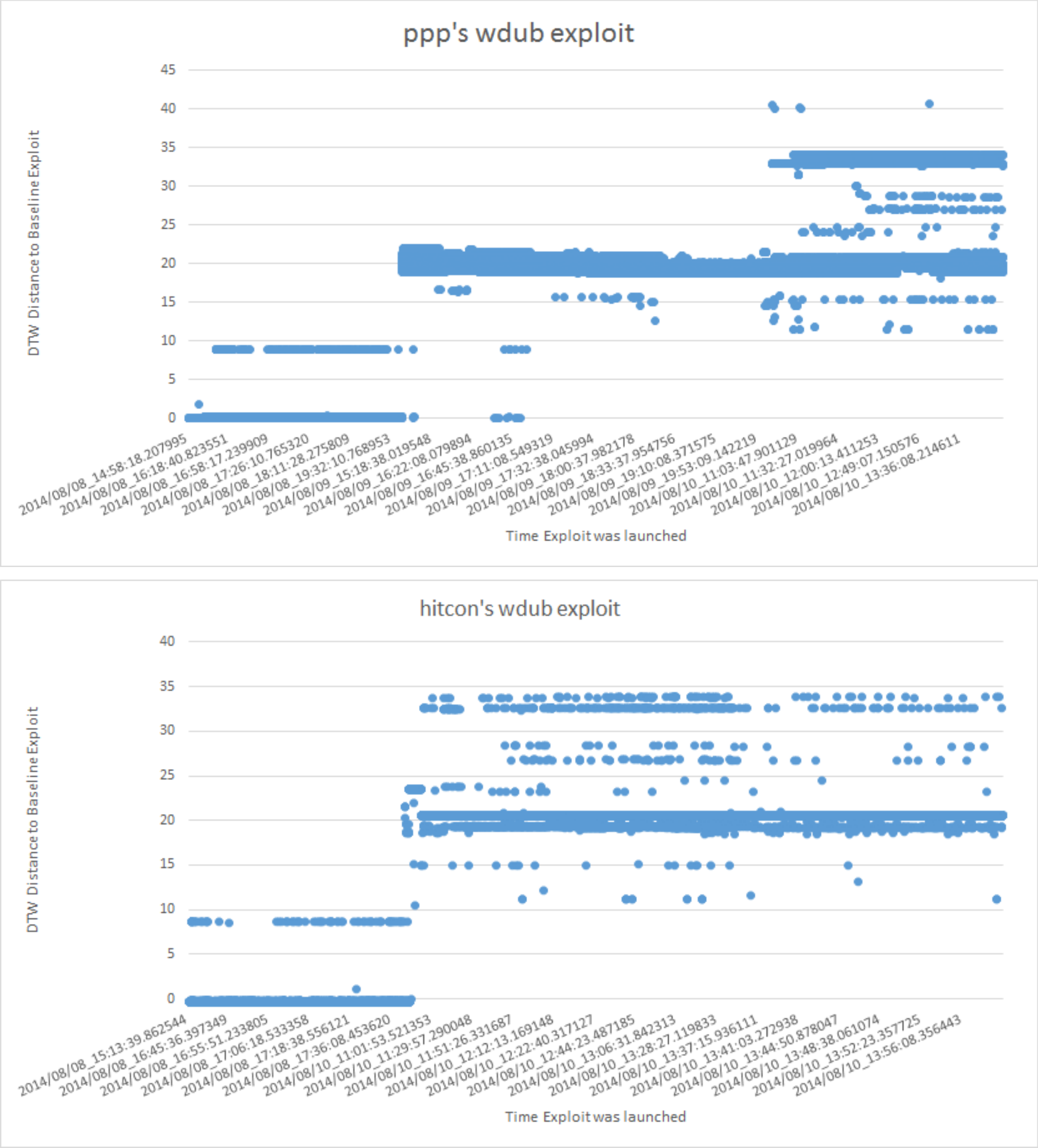
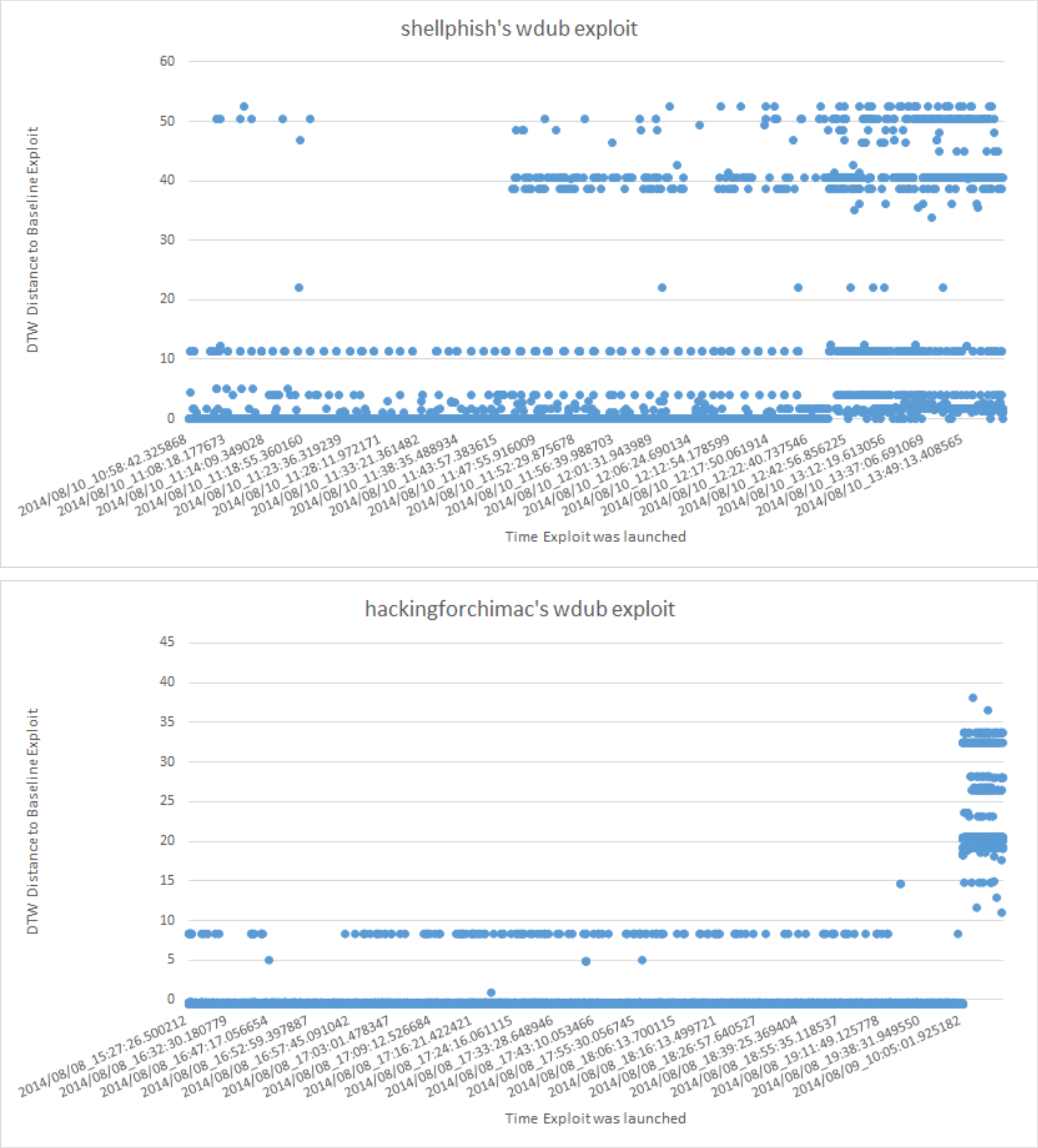


Figure 4.7: Variants of shellphish's and hackingforchimac's wdub exploit



We could not draw any conclusions on whether the number of variants and how teams used these variants (e.g., using them simultaneously or sequentially) affected a team's standing as we believe there were other factors that affected exploit polymorphism. For example, both competent and weak teams captured and reused the exploits of other teams, which resulted in them having multiple variants. A area of future work could be to study the polymorphism of only exploits that are not reused. This would give us a better idea whether morphing their exploits would result in teams having a better standing.

4.2.2 Teams' Defensive Capabilities

In this subsection, we investigate how much a team's defensive capabilities contribute to its success in the CTF competition. The metrics we used to determine defensive capabilities are:

- Number of tokens lost
- Percentage of tokens lost that were found on wire during ex-filtration
- Reuse of exploits
- Average time to patch a service
- Service uptime

Number of Lost Tokens

Analogous to using the number of captured tokens as an indicator of a team's offensive capability, the number of tokens stolen from a team was a direct indicator of how well that team defended their services. Information on the number of tokens a team had lost was acquired from the game state database. Each team may have lost its token more than once, since a team's services may have been exploited by more than one team. Table 4.11 shows the number of tokens a team had lost to all other teams.

Table 4.11: Table showing the number of tokens stolen from each team

Team	No. of Tokens Lost (1 count for each token lost to each team)
ppp	360
hitcon	505
dragonsector	932
reckless	607
blue-lotus	270
mmibh	943
raon_asrt	786
stratum	686
team9447	827
kaist	870
routards	672
mslc	1814
binja	16
codered	1082
w3stormz	1094
penthackon	744
balalaikacr3w	2802
gallopsled	1697
shellphish	331
hackingforchimac	651

From the table, we observe that teams that lose the least tokens do not necessarily rank better. *binja* for example lost only 16 tokens but ranked somewhere in the middle. Similarly, *dragonsector* lost more tokens than the bottom two teams (*shellphish* and *hackingforchimac*) yet performed better than both of them. Recall that the scoring system works in a way that flags are distributed among all teams that have successfully compromised a service and redeemed its token. Therefore, the effect of losing the tokens could possibly be distributed across many teams, not giving any one team a significant lead. In addition, the number of tokens lost can be offset by how many tokens a team manages to capture. Hence, the number of tokens lost would not have a direct impact on a team's final ranking.

Percentage of Tokens Stolen that were Found on Wire During Ex-filtration

In addition to how many tokens were lost by a team, we also looked at what percentage of those stolen tokens were ex-filtrated in the clear (or trivially encoded using the Base64 encoding scheme). Teams losing tokens have a higher chance of discovering and reusing their attacker's exploit if they were able to detect the actual token ex-filtration events. Table 4.12 shows the number of lost tokens that were visible in the network traffic captures.

Table 4.12: Table showing the number of lost tokens seen on the wire

Team	Rank	No. Of Tokens Lost	No. Of Lost Tokens Found (In Clear)	No. Of Lost Tokens Found (Base64)	No. Of Tokens Not Accounted	Percentage Of Tokens Found
penthackon	16	744	170	11	563	24.33%
ppp	1	360	227	7	126	65.00%
hackingforchimac	11	651	444	50	157	75.88%
hitcon	9	505	380	7	118	76.63%
shellphish	7	331	272	0	59	82.18%
blue-lotus	10	270	222	1	47	82.59%
raon_asrt	5	786	638	40	108	86.26%
team9447	2	827	742	0	85	89.72%
routards	4	672	587	20	65	90.33%
stratum	17	686	607	13	66	90.38%
reckless	3	607	547	10	50	91.76%
kaist	6	870	739	71	60	93.10%
dragonsector	15	932	852	20	60	93.56%
mmibh	12	943	848	47	48	94.91%
gallopsled	18	1697	1549	81	67	96.05%
mslc	14	1814	1700	46	68	96.25%
w3stormz	13	1094	1045	9	40	96.34%
codered	8	1082	1040	7	35	96.77%
balalaikacr3w	19	2802	2593	137	72	97.43%
binja	20	16	16	0	0	100.00%

We see from the table the the teams that had most of their tokens ex-filtrated in the clear are not necessarily the team that ranked the best. *ppp*, who was ranked first, had 65% of their lost tokens ex-filtrated in the clear, while *hackingforchimac*, who was ranked last had more of their lost tokens ex-filtrated in the clear. It could mean that teams that had a higher number lost tokens ex-filtrated in the clear, yet ranked badly had low competency in capturing and reusing exploits.

Reuse of Exploits

When a team is unable to find a vulnerability and/or develop an exploit for a vulnerable service, a strategy they may turn to is to capture exploits that the other teams are using against them, and repackage them as their own.

We wrote a script to extract the first successful exploit for each team for each service (i.e., the first exploit that was launched that successfully stole a token). The script worked based on the heuristics that if we saw a token on the network, then the associated packets preceding it contained the exploit. This heuristic allowed us to extract all but one of the "first exploit" sent by each team. The exploit that we failed to extract was *blue-lotus'* exploit for the imap service. From manually inspection, we discovered that the "exploit" we had extracted was a series of shell commands (such as ls, cat flag), which indicated that the exploit came sometime before. We had to rely on manually searching the network traffic to find *blue-lotus'* imap exploit.

In order to determine whether teams reused exploits from other teams, we performed pair-wise comparison of every team's exploit with every other team's to measure the difference between each exploit pair for a given service. For each service, the score was calculated by finding the distance between the two exploits (using Dynamic Time Warping code from [36]) being compared and normalizing by dividing by the median exploit size of the all the teams for that service. We then plotted the scores on a scatter plot as shown in Figures 4.8, 4.9, 4.10 and 4.11. These scatter plots are sorted by their DTW distance. The x-axis gives the team pairs and the y-axis gives the corresponding normalized DTW distance. Teams missing from the scatter plot failed to redeem any tokens for the service, from which we concluded they failed to exploit the service.

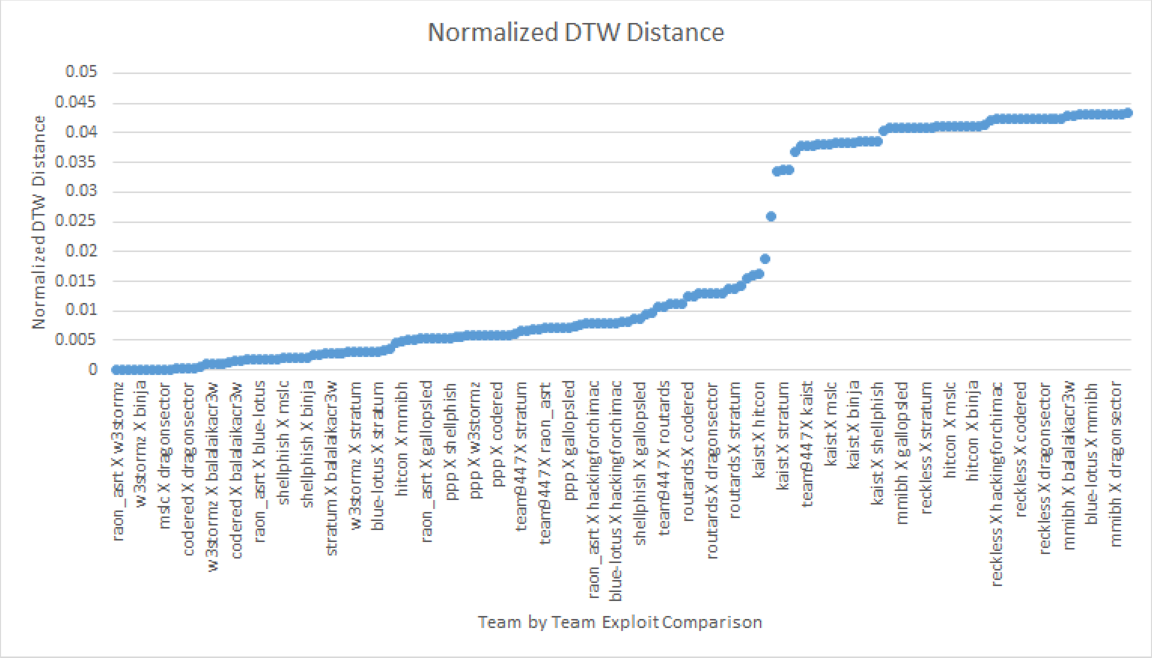


Figure 4.8: Scatter Plot showing normalized DTW distance for pairwise comparison of eliza exploit

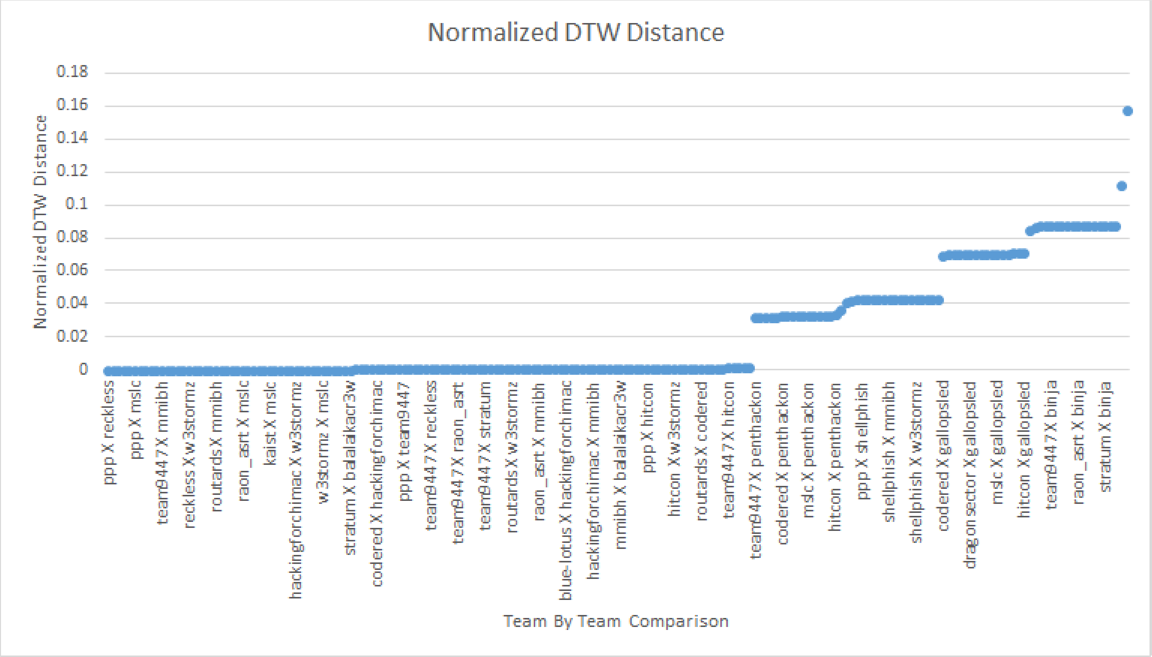


Figure 4.9: Scatter Plot showing normalized DTW distance for pairwise comparison of wdub exploit

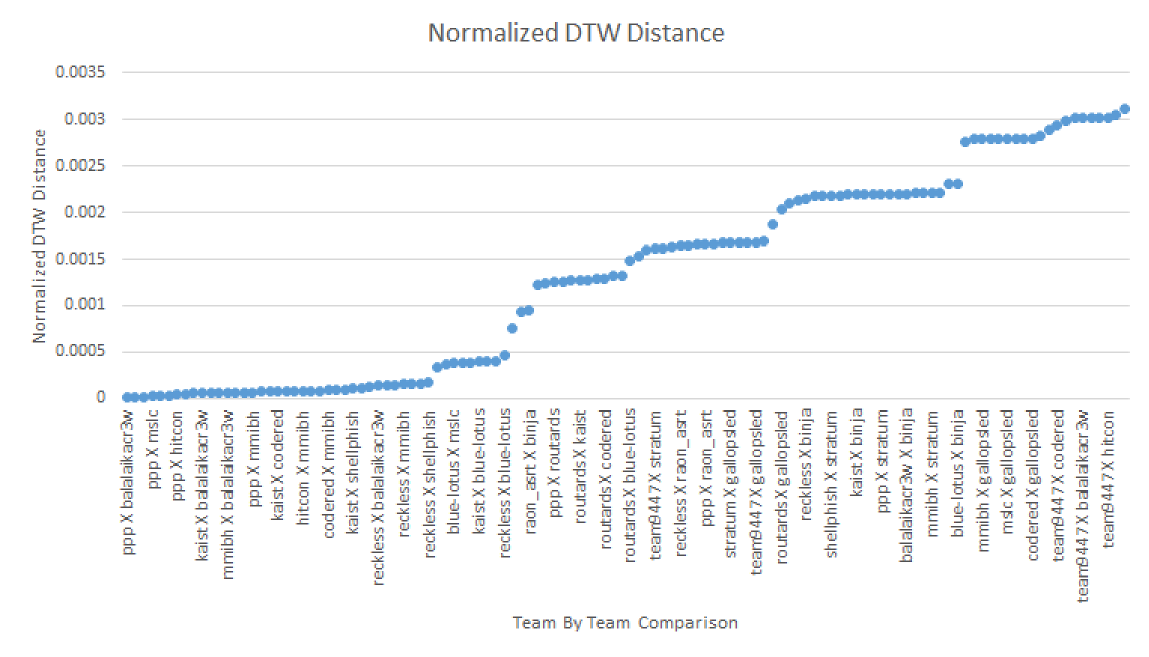


Figure 4.10: Scatter Plot showing normalized DTW distance for pairwise comparison of justify exploit

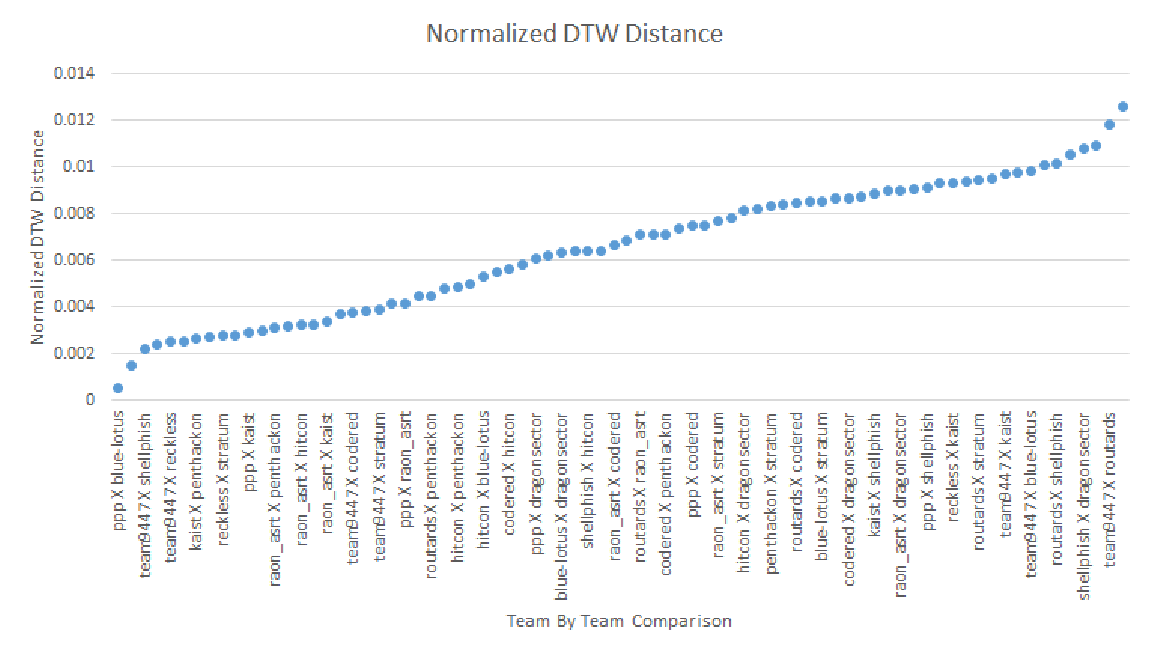


Figure 4.11: Scatter Plot showing normalized DTW distance for pairwise comparison of imap exploit

Inspecting the scatter plots, we can see:

- **There are clear threshold values we can use to identify exploits that are similar.** On the scatter plots, we should only look at the group of dots that have normalized DTW distance ranging from 0 to the first gap in the linear group of dots. For example, in Figure 4.8, we can clearly see a gap at a value of 0.02. Hence, for eliza, we consider exploits similar if they have a normalized DTW distanace of 0.2 or less.
- **The groups of linear dots indicate variants.** Looking at the scatter plots, we can see isolated groups (except for the imap exploit). These isolated groups indicate that there are a number of exploits that have equal DTW distance from a reference exploit and therefore show the existence of variants.

Except for the imap exploit's scatter plot shown in Figure 4.11, we can clearly see groupings, indicating exploit reuse. For the imap exploit, no such groupings appeared. Manually inspecting the imap exploit, we realized that the nature of the exploit did not lend itself to methods that used byte-value comparisons (like DTW) because the exploit was text-based and the text is changed between exploit instances. Therefore, to determine similarity and variants, we resorted to manual inspection. We determined that there were two variants to the imap exploit, as shown in Table 4.13. One used the IMAP commands in the sequence LOGIN, CREATE, SELECT and the other variant used the sequence, LOGIN, CREATE, APPEND, FETCH.

Table 4.13: Table showing the two imap exploit variants (partial)

Variant A	Variant B
a049 xresuwen uokq cuvl	000b XRESUWEN 791858c5c8779...
a050 login uokq cuvl	000c LOGIN 791858c5c8779...
a051 create 1111111...	CREATE a
AAAAA	000e SELECT ////////////...
a053 fetch 11111111...	...
a054 select 1111111...	...

By comparing all the team's exploits to a baseline exploit (we used *ppp's* exploit as the baseline because they won the CTF and they had exploits for every service), we can clearly see which team's exploits were similar (with the exception of imap). Figures 4.12, 4.13, and 4.14 show the number of variants for each exploit type, and we compiled the exploit variant information in Table 4.14. The teams missing from any of the figures and the table are teams that failed to exploit that service.

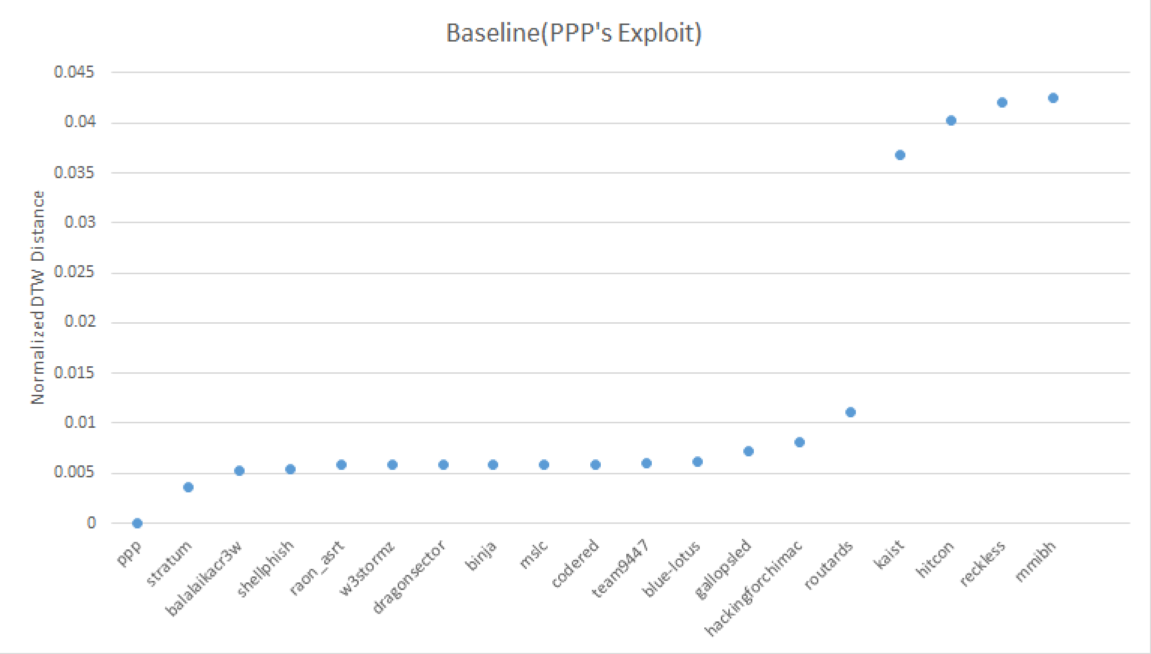


Figure 4.12: Scatter Plot showing groups of variants for eliza exploit

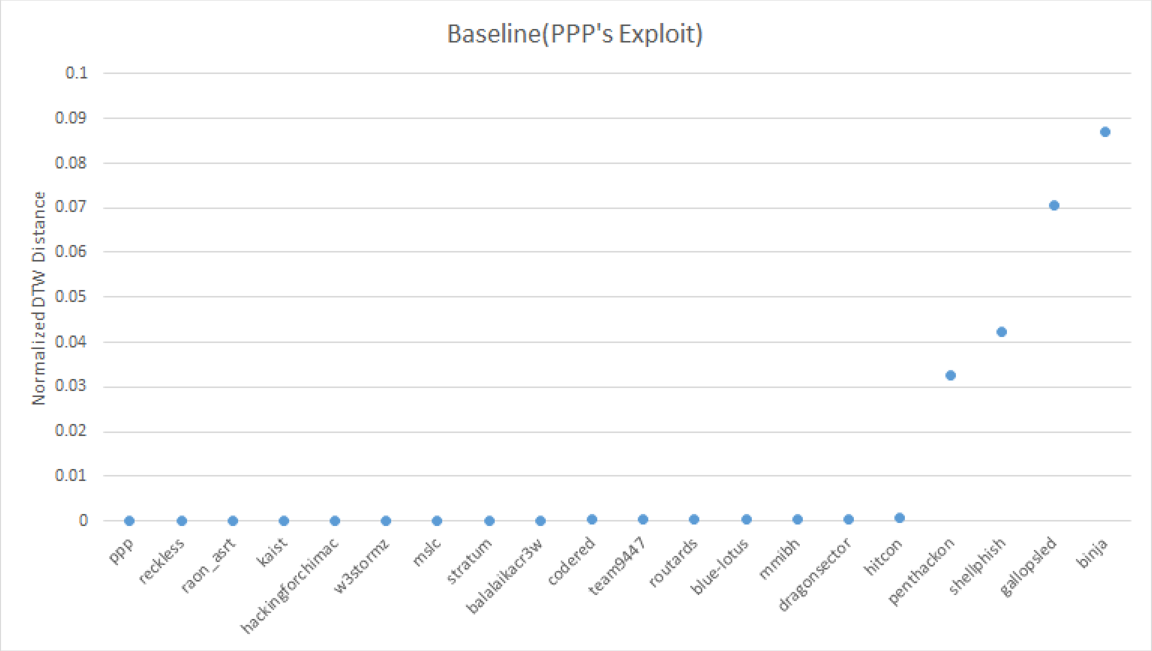


Figure 4.13: Scatter Plot showing groups of variants for wdub exploit

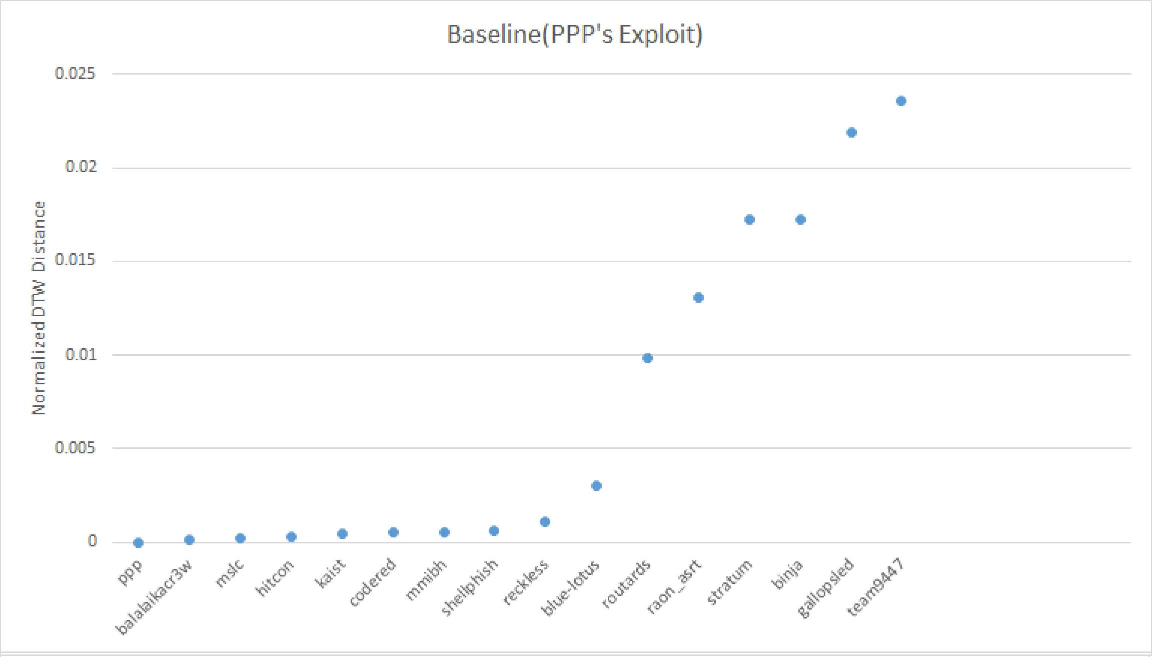


Figure 4.14: Scatter Plot showing groups of variants for justify exploit

Table 4.14 shows the number of variants we found for each exploit that targeted each service.

It also shows the teams that used similar variants. From the table, we see that eliza had two variants, A and B, while wdub, justify and imap had three, three and two variants, respectively. What is interesting to note is that there were variants used by more than one team, indicating exploit reuse. For example, variant B of the eliza exploit was used by *reckless, hitcon, kaist* and *mmibh*, whereas variant A was used by the rest of the teams. This implies that teams were not taking sufficient measures (if at all) to prevent their exploits from being captured and reused.

From Table 4.6 that shows the exploit development time of each service, we know that *hitcon* was the first team to develop the exploits for eliza, wdub and imap, and *ppp* was the first team to develop the exploit for justify. Therefore, comparing that with Table 4.14, we deduces that for eliza's variant A exploit, *ppp's* exploit was most likely captured and reused. Likewise, for eliza's variant B exploit, *hitcon's* exploit was the one that was most likely captured and reused. For wdub and imap, it is likely that *hitcon's* exploits were also captured and reused since *hitcon* was the first team to develop the exploits, and many teams were found to be using the same variants as *hitcon*.

Table 4.14: Table showing the number of exploit variants and their use by each team

Exploit	Variants	Teams Using Variant
eliza	A	ppp, team9447, routards, raon_asrt shellphish, codered, blue-lotus hackingforchimac, w3stormz, mslc dragonsector, stratum, gallopsled balalaikacr3w, binja
	B	reckless, hitcon, kaist, mmibh
wdub	A	ppp, team9447, reckless, routards, raon_asrt ,codered, blue-lotus, hackingforchimac, w3stormz, mslc dragonsector, stratum, hitcon, balalaikacr3w, kaist, mmibh
	B	binja, gallopsled
	C	shellphish, penhackton
justify	A	ppp, reckless, shellphish, kaist, codered, hitcon, blue-lotus, mmibh, mslc, balalaikacr3w
	B	raon_asrt, routards
	C	stratum, binja
	D	gallopsled, team9447
imap	A	ppp, reckless, routards, kaist, codered, hitcon, blue-lotus, dragonsector, stratum
	B	raon_asrt, shellphish,team9447, penthackton

Average Time to Patch

Time to Patch is a metric that measures how fast on average (in terms of rounds) a team was able to patch its vulnerable services. We made the assumption that for a given round, if a team lost a token for a particular service, then the service was exploitable/unpatched. A service was considered patched in round N, if for all rounds greater than or equal to N, the service lost no tokens. In this way, we measured Time to Patch using game state data as a proxy rather than manually inspecting the network traffic to determine whether a service has indeed been patched. While this method was not full proof, it did offer a simple and efficient way to measure Time to Patch. Note that if a team installed a backdoor on a server it exploited, then they could still retrieve and redeem tokens even though the service they had previously exploited had subsequently been patched. In such cases, the database showed that tokens were redeemed even though the service was patched. Follow-on research work could improve this metric by attempting to use network traffic to determine when a service was patched. One could look for cases in the network traffic where an exploit that has been known to work in one round (as evidenced by its ability to steal tokens) but fails subsequently.

It is important to note that in calculating the average time to patch, we only considered the four services, namely eliza, wdub, justify and imap. The badger service was omitted since only one team was shown to have redeemed a token for it. Therefore the badger service was considered an outlier. Also note that each service had more than one vulnerability. Hence, even if a team had patched one of the vulnerabilities, it could still have been exploited. Table 4.15 shows the rounds in each team's service that were patched.

Note that the following team's services never had a token stolen from them. The game state data indicated that these services were up intermittently, and we could not determine whether these services were patched. Hence, we considered these services to be unpatched for the duration of the event, and we assigned a value of 273 as their Time to Patch value:

- team binja's wdub service
- team binja's justify service
- team binja's imap service
- team stratum's imap service
- team balalaikacr3w's imap service
- team gallopsled's imap service

Table 4.15: Table showing the time (in rounds) each team's service was patched

	Time to Patch (in Rounds)				
Team	eliza	wdub	justify	imap	Average
ppp	250	273	237	112	218
hitcon	132	273	224	159	197
dragonsector	220	273	193	113	199.75
reckless	228	268	273	119	222
blue-lotus	174	270	196	113	188.25
mmibh	186	273	224	273	239
raon_asrt	266	273	211	273	255.75
stratum	268	273	202	273	254
team9447	189	273	225	173	215
kaist	227	273	273	272	261.25
routards	273	273	226	273	261.25
mslc	271	273	273	112	232.25
binja	259	273	273	273	269.5
codered	239	273	273	154	234.75
w3stormz	270	273	240	247	257.5
penthackon	273	273	273	254	268.25
balalaikacr3w	261	273	273	273	270
gallopsled	273	273	273	273	273
shellphish	245	273	221	206	236.25
hackingforchimac	272	273	242	273	265

From the table, we can observe the following:

- The top five teams were able to patch their services before the 224th round. Conversely, four of the bottom five teams only patched their services after the 265th round.
- The top five teams were able to patch at least one of their services before the 136th

round, which was the midpoint of the CTF. Conversely, all of the bottom five teams did not patch any of their services until after the 206th round.

We expected correlation between the average time to patch and a team's final standing. In addition, a team's ability to analyze a service and find its vulnerability had bearing on its ability to patch it, since before a team could patch the service, it first had to find the vulnerability.

Service Level Agreement

The Service Level Agreement metric measured the percentage of the 272 rounds that a team's services were up and judged to be functioning correctly. The game state database had information on all the penalties incurred by each team for having a service go down. The maximum uptime a team may have occurred when all services were up for all rounds, which would be 1360 (272 rounds multiplied by 5 services). Hence, the service level agreement fulfilment ratio (SLA) for each team was calculated as follows:

$$SLA = (1 - (penalties/max.uptime(inrounds))$$

Table 4.16 shows the Service Level Agreement fulfilment ratio of each team. We see that teams who were ranked higher (e.g., *ppp, dragonsector, reckless* and *blue-lotus*) had higher SLA values than teams who were ranked lower. Therefore, we believe there is a strong correlation between keeping a service up and a team's final standing.

Table 4.16: Table showing the Service Level Agreement fulfilment each team

Team	Num. of Penalties	SLA
ppp	145	0.893382
hitcon	67	0.950735
dragonsector	82	0.939706
reckless	86	0.936765
blue-lotus	111	0.918382
mmibh	69	0.949265
raon_asrt	134	0.901471
stratum	166	0.877941
team9447	117	0.913971
kaist	73	0.946324
routards	171	0.874265
mslc	137	0.899265
binja	272	0.8
codered	208	0.847059
w3stormz	211	0.844853
penthackon	174	0.872059
balalaikacr3w	157	0.884559
gallopsled	180	0.867647
shellphish	171	0.874265
hackingforchimac	254	0.813235

4.3 Analysis of Findings

We used the Pearson product-moment correlation coefficient to calculate how much each quantitative metric was correlated to a team's final standing. The Pearson coefficient is a value ranging from -1 to +1 and tells us how correlated two series are. -1 implies a negative correlation and +1 implies a positive correlation. 0 implies no correlation [37].

Table 4.17: Table showing the Pearson product-moment correlation coefficient scores of each quantitative metric

Metric	Pearson Coefficient Scores
Number of Tokens Redeemed by Each Team	-0.72851
Percentage of Redeemed Tokens Seen in Ex-filtration Traffic	0.211748
Number of Tokens Lost By Each Team	0.361034
Percentage of Stolen Tokens Seen in Ex-filtration Traffic	-0.02029
Average Time To Develop an Exploit	0.723385
Rate of Exploitation	0.173
Number of Attacking Hosts Used	0.4738
Number of Callback Ports Used	0.26404
Average Time to Patch	0.716801
Service Level Agreement	-0.70693

In addition to studying the quantitative metrics, we also looked at qualitative metrics such as the ex-filtration techniques, variation of attack parameters, and reuse of exploits. Based on the metrics we have discussed, we made the following observations and analysis:

1. **Attacking first mattered.** Both the number of tokens redeemed and the average time to develop an exploit had relatively high correlation to a team's final ranking. This observation was expected. The earlier a team was able to successfully attack a service, the longer they had to capture tokens and accrue points. This observation also indicates that a skilful team should perform better since the more skilful a team is, the faster we expect them to discover and exploit the vulnerabilities present in the service.
2. **Obfuscating ex-filtration traffic did not provide added benefit.** The percentage of tokens seen in the network traffic had low correlation to a team's final standing. Of the top five teams, three of them (*ppp*, *dragonsector* and *blue-lotus*) ex-filtrated over 99% of their tokens in the clear. Therefore, it appear that obfuscating ex-filtration traffic did not influence a team's final standing in this CTF. However, one could argue

that the benefits of obfuscating ex-filtration traffic and exploits was not apparent in this CTF. In this CTF, most of the teams sent their exploits without obfuscation and also ex-filtrated most of their tokens in the clear. This practice may have given less skilful teams (who were unable to develop an exploit on their own) a chance to capture exploits off the network and use them to steal tokens when they would otherwise not be able to. Hence, it may be the case that less skilful teams were able to achieve a better standing simply because teams did not protect their exploits and/or ex-filtration traffic.

3. **The method of ex-filtration did not matter.** We looked at two payloads that the teams used—session-reuse payloads and callback payloads. Comparing teams that used only one type of payload with teams that use both, we did not find any correlation between the type of ex-filtration techniques used to a team's final standing. We found that half of the teams of various standings used both types of payloads and the other half used only session-reuse payloads. In this CTF, most of the payloads we saw were straight forward shell command execution payloads. It seemed that teams did not require sophisticated payloads to be able to steal tokens, and therefore they used the most simple and efficient payload they had.
4. **Exploiting a service multiple times per round had no added benefit.** The Pearson coefficient score for rate of exploitation is low, suggesting that whether one used automation to attack rapidly, or with a more modest frequency, did not have a large effect on a team's final standing. We were expecting teams to avoid launching exploits indiscriminately to reduce the chances of their exploits being captured. However, we saw that teams were launching exploits multiple times per round. The top team *ppp*, had an exploitation rate of 309.8 exploits per round, which was equivalent to sending 3.8 exploits to each service of each team. Perhaps teams were not aware of when each round begins and ends, and therefore launched their exploits continuously to avoid missing the opportunity to score points in a round. Or perhaps teams were not concerned so much about their exploits being captured as they were about scoring points.
5. **Patching was important, but not as important as keeping services operational.** The Pearson Coefficient score for SLA was high, whereas the Pearson Coefficient score for Tokens Stolen from a team was relatively low. Having a service go down for a round would mean losing all 19 flags associated with that service for that round,

which was costly. Having an unpatched service up did not necessarily mean the service would be exploited. Hence, it would make more sense to have a service up, even though it was unpatched than to have it go down. In addition, the consequence of having a service exploited may not have been as detrimental to a team's standing as it appeared since the effect of losing tokens may have been distributed across many teams, thus not giving any one team a significant advantage. From an attacker's perspective, bringing a service down could be an effective strategy. In cases where developing an exploit that steals a token is more difficult than developing a lesser one that simply crashed the service, it may be viable for a team to simply use the lesser exploit. Crashing an opponents service would not necessarily gain a team any flags, but it would cause the opposing teams to lose flags.

6. **There is evidence to indicate that some teams had a higher situational awareness than others.** Studying the scatter plots of polymorphic exploits, we saw that the top two teams switched to different exploits sometime during the CTF event. Whereas the bottom ranked team did not. As a result, the two top teams were able to exploit most of teams for a longer duration that the bottom ranked team. This indicates that the top two teams had good awareness of what was happening and adjusted their actions accordingly.
7. **There is an interplay of factors other than tokens redeemed, tokens lost, and SLA.** Redeeming more tokens did not necessarily mean a higher final ranking. *ppp* redeemed the most tokens at 2411 and was ranked first, but *raon_asrt* who redeemed the second most number of tokens was only ranked 7th. Likewise, *binja*, who redeemed the least number of tokens at 234, was not placed last but 13th. In a similar vein, losing the most tokens does not necessarily mean a low scorer. *raon_asrt* lost 786 flags, which is more flags than the 16 lost by *binja*, but *raon_asrt* was still ranked higher. This hints at some interplay between factors other than tokens redeemed, tokens lost, and SLA. This implies that an effective strategy cannot be simply one that focuses purely on attack, defence or uptime, but has to be mindful also of the scoring and penalising mechanisms of the CTF. Recall that for each round, flags of a service were equally distributed among all the teams that had successfully stolen a token from that service. Therefore, simply attacking services that had been exploited frequently would not yield as large a share of flags than attacking services that had been exploited less frequently.

8. **Protecting exploits from capture and reuse would only be useful if everyone participated.** From our data, we see that the exploits used be each team are similar to exploits used by other teams. This implies that teams were reusing exploits from other teams. Protecting exploits from capture and reuse would incur extra effort that may not pay off. In scenarios like this CTF, where majority of the teams were not obfuscating their exploits, being the sole team to obfuscate its exploit would bring no benefit. Teams that were unable to capture and reuse the exploit of one team because the exploits were obfuscated, could choose to capture and reuse the exploits of the other teams that did not obfuscate their exploits. Hence, instead of investing effort into developing exploits that are hard to capture or analyze, it may be more profitable for teams to spend time finding new vulnerabilities and developing exploits for them.
9. **A purely offensive approach may have been more effective than an approach that balances offense and defense.** Since the number of tokens a team lost had low correlation to its final standing, whereas the number of tokens a team managed to steal had a high correlation, it stands to reason that resources put into attacking would be more beneficial than resources dedicated to patching. This would affect on how one should make up a team to compete in the CTF. Based on our observations, it stands to reason that a team that had a majority of its members skilled at offense may perform better than a team who had a majority of its members skilled at defense.

THIS PAGE INTENTIONALLY LEFT BLANK

CHAPTER 5: CONCLUSION

In this thesis, we studied DEF CON 22 CTF traffic and game state data in an attempt to learn strategies employed by the participating teams as well as to discover factors that correlated a team's skill to its overall standing in the CTF event.

In order to efficiently process the large amounts of network packet captures, we used the following methodology:

1. Processed network traffic (that contains the full payload) into a light-weight representation so that we could more efficiently visualize and generate statistics.
2. Used data visualization techniques to help zoom in to areas of interest.
3. Correlated features and observables network data with game state data of scoring infrastructure to efficiently narrow down traffic and behaviours of interest.
4. Adopted the heuristics of using one successful exploit from each team as representative for every instance of a team's exploit for a given service. This saved us the work of having to inspect every instance of every exploit from each team and reduced it to simply inspecting one exploit instance for every exploit from each team.

We created quantitative and qualitative metrics, and we studied how each correlated to a team's final standing in the event. For quantitative metrics, we used the Pearson product-moment correlation coefficient. The metrics included:

- Number of tokens redeemed
- Percentage of tokens found on wire during ex-filtration
- Average time to develop an exploit
- Use of publicly available payloads
- Exfiltration techniques
- Rate of exploitation
- Variation of attack parameters
- Exploit polymorphism
- Number of tokens lost

- Percentage of tokens lost that were found on wire during ex-filtration
- Reuse of exploits
- Average time to patch a service
- Service uptime

We learned that:

- **Teams automated their attacks to exploit other teams at regular intervals.** From the network traffic, we saw regular spikes corresponding to exploit traffic. This indicated that the teams were using some form of automation to launch their attacks.
- **Teams actively scanned the host systems of the other teams and attempted to exploit them.** We saw evidence that teams were scanning address ranges of other teams in an attempt to find active hosts. We observed an instance where one of the teams tried to exploit a phpMyAdmin interface running on one of the other team's client PCs.
- **There was a high correlation, as indicated by the Pearson Coefficient score, between the number of tokens redeemed, the average time to develop an exploit and a team's final standing.** This implied that attacking was important and being able to find a vulnerability, and exploit it as early in competition as possible, had benefit. This was expected, as the earlier in the competition one developed an exploit, the more time one had to use it and accrue points.
- **Obfuscating ex-filtration traffic did not provide added benefit.** The percentage of tokens seen in the network traffic had low correlation to a team's final standing. Electing not to obfuscate ex-filtration traffic did not effect a team's final standing. In fact, three of the top five teams in the competition ex-filtrated over 99% of their tokens in the clear.
- **Method of ex-filtration did not matter.** We looked at two payloads that the teams used—session-reuse payloads and callback payloads. Comparing teams that used only one type of payload with teams that used both, we do not find any correlation between the type of ex-filtration techniques used to a team's final standing. We found half of the teams of various standings using both type of payloads and the other half using only session-reuse payloads.
- **Exploiting a service multiple times per round had no added benefit.** We saw that most teams exploited each service multiple times per round, even though exploiting a

service once successfully was sufficient to steal a token. From the Pearson coefficient scores, we saw little correlation between how many exploits were launched per round versus how the teams scored in the end.

- **Patching was important, but not as important as keeping services operational.** We saw that maintaining a high Service Level Agreement ratio had a high correlation with a team's final standing, whereas there was a low correlation between tokens lost and a team's final standing.
- **There was evidence to indicate that some teams had a higher situational awareness than others.** Studying the scatter plots of polymorphic exploits, we saw that the top two teams switched to different exploits sometime during the CTF event. Whereas the bottom ranked team did not. As a result, the two top teams were able to exploit most of teams for a longer duration that the bottom ranked team. This indicates that the top two teams had good awareness of what was happening and adjusted their actions accordingly.
- **There was an interplay of factors other than tokens redeemed, tokens lost, and SLA.** Redeeming more tokens did not necessarily mean a higher final ranking. *ppp* redeemed the most tokens at 2411 and was ranked first, but *raon_asrt*, who redeemed the second most number of tokens was only ranked 7th. Likewise, *binja*, who redeemed the least number of tokens at 234, was not placed last but 13th. In a similar vein, losing the most tokens did not necessarily mean a low scorer. *raon_asrt* lost 786 flags, which is more flags than the 16 lost by *binja*, but *raon_asrt* was still ranked higher. This hints at some interplay between factors other than tokens redeemed, tokens lost, and SLA.
- **Protecting exploits from capture and reuse would only be useful if everyone participated.** From our data, we see that the exploits used be each team are similar to exploits used by other teams. This implies that teams were reusing exploits from other teams. Protecting exploits from capture and reuse would incur extra effort that may not pay off. In scenarios like this CTF, where majority of the teams were not obfuscating their exploits, being the sole team to obfuscate its exploit would bring no benefit. Teams that were unable to capture and reuse the exploit of one team because the exploits were obfuscated, could choose to capture and reuse the exploits of the other teams that did not obfuscate their exploits. Hence, instead of investing effort into developing exploits that are hard to capture or analyze, it may be more profitable

for teams to spend time finding new vulnerabilities and developing exploits for them.

- **A purely offensive approach was more effective than an approach that balanced offense and defense.** Since the number of tokens a team lost has low correlation to its final standing whereas the number of tokens a team managed to steal had a high correlation, it stands to reason that resources put into attacking were more beneficial than resources dedicated to patching.

From what we have observed from the data, we found the following to be surprising:

- Teams were aggressive in launching their exploits. We expected that teams would be judicious in their rate of exploitation so as to reduce the chances of their attacks being discovered and their exploit captured. However, what we observed was that teams continuously launched their exploits throughout the duration of the game. Presumably, they were more concerned about stealing tokens than they were about protecting their exploits from capture.
- Few teams bothered with obfuscating their exploits and their ex-filtration traffic. Similar to the point above, it appeared that teams were not concern with hiding their attacks or indicators of compromise. We had expected that teams would operate in a more stealthy manner and attempt to make it more difficult for other teams to detect their offensive activities.
- Teams used relatively simple exploits and payloads. Related to the above point, we did not see evidence that teams obfuscated their exploits nor used sophisticated payloads. All the payloads that we found on the network were relatively simple that ran shell commands to read the token file, or created a reverse connection to send the token file back to the attacker. Some of the teams had experienced and skilful members that had won several CTFs previously; hence, we expected their attacks to have had a higher level of sophistication.
- A purely offensive approach may be more effective than one that balances between offense and defense. An attack-defense type CTF should take into account both the offensive and defensive abilities of the participating teams. However, from our findings, it appeared that an offensive-biased strategy may be a more effective one that a strategy that balanced offense and defense.

5.1 Future Work

There are still a number of areas that warrant further study:

Investigate Collusion Between Teams

We noted communication between host machines of two competing teams that did not appear to be attack traffic. We were unable to determine the nature of this communication. For the purposes of understanding how feasible collusion can be as a strategy, it would be interesting to study whether teams are colluding (either over the CTF network or out-of-band), how they are colluding, and how much does collusion contribute to their performance. Future work could look at the kinds of collusion and study whether teams trade/share exploits or flags.

Implement Better Techniques for Measuring Payload and Exploit Differences

We chose to utilize Dynamic Time Warping (DTW) for measuring distance between exploits and payloads. However, this did not work well for comparing the exploits for the imap service or for comparing publicly available shellcodes. DTW focused on the value of each byte and its relation to other bytes, whereas a better technique might have been to measure the semantic differences between two payloads. A better technique to measure payload and exploit differences would have allowed us to make better assessment on whether teams used publicly available payloads. It would also have allowed us to be more accurate in determining the number of exploit variants used in the CTF.

Compare Exploit Payloads Against a Larger Corpus of Public Shellcodes

For this thesis, we only compared each team's exploit against four publicly available shellcodes taken from shell-storm.org and the Metasploit Framework. This should be expanded to include more shellcodes from more repositories. It would be interesting to know whether utilizing publicly available exploits was a worthwhile strategy. Intuitively, we would expect DEF CON CTF participants to use custom payloads, since the architecture and execution environments may be customised in these CTFs, so that public payloads would not work. It would be surprising to discover teams that use and even perform well in DEF CON CTFs using only publicly available payloads.

Study the Use of Backdoors

A interesting area of future work would be to determine whether teams install backdoors (as opposed to simple shell scripts) in the systems they exploit and study if the utilization of such tactics helps a team to perform better. Additionally, it would be interesting to categorize the types of backdoors used and how that too affects a team's ranking. Such a study would help in developing a strategy on how backdoors can be effectively used to enhance a team's performance.

Study the Time to Turn Around Exploits

A metric to measure the defensive capabilities of a team might be to examine how quickly a team can repurpose a captured exploit to suit its own needs. Teams that want to prevent other teams from reusing their exploit would then attempt to obfuscate their exploits to frustrate any reverse-engineering attempts. It would be interesting to know whether purely reusing exploits (as opposed to finding the vulnerabilities and developing their own exploit) would be a worthwhile strategy, and if so, what turnaround time would be required for the strategy to be effective.

Study the Employment of Chaffing or Similar Techniques

As mentioned, teams monitored their network traffic in hopes of finding an exploit they could reuse. Obfuscating attack traffic may not always be possible, so teams may employ chaffing instead—which involves flooding the network of their target before sending their exploit. The desired effect of chaffing is to overwhelm the target with network traffic in order to hide the actual exploit in the resulting noise. It would be interesting to learn what kind of chaffing techniques that teams employ and how effect each of the techniques can be in positively affecting the performance of a team.

Analyze the Rationale for Exploit Polymorphism

From our findings, we know that teams polymorph their exploits. However, what we do not yet know, and therefore should investigate, would be:

- Whether teams developed a new variant for an exploit in response to key events. For example, do teams develop variants upon learning that their exploits were captured and re-used? If so, how do teams learn that fact?

- Whether the use of variants was guided by some underlying strategy. For example, do teams develop a number of variants at once but use them only over a period of time?

Study the Re-use of Exploits from a Temporal Perspective

Our findings show that team do indeed capture and re-use the exploits of other teams. However, we did not study how these captured exploits propagated to other teams. For example, *ppp* was the first to develop the exploit for justify, and this exploit was subsequently re-used by a number of teams. It would have been interesting to find you:

- Who was the first team that captured *ppp's* exploit?
- Did the other teams that used *ppp's* exploit also captured it from *ppp*, or did they capture it from another team that reused the exploit.

In addition, it would be interesting to test our hypothesis that teams discovered exploits by tracing back traffic that contained leaked tokens. It is plausible to believe that not all exploits that a team launched were successful. In such cases, did the teams on the receiving end of failed exploits detect these failed exploits and used these failed exploits as blueprints to develop their own?

Compare Study Findings Across Multiple DEF CON CTFs

Currently we only studied the data from one DEF CON CTF event. Hence, all our observations and findings only pertain to the DEF CON 22 CTF. As future work, we should study other CTFs to and test whether our findings and hypothesis hold across other DEF CON CTFs.

Study CTF Dynamics and How it Affects Game Play

Other iterations of DEF CON CTFs would have different variety of services and architectures as well as different number of participating teams of various competency levels. As future work, we could look at how different services, architectures, participant composition and their resulting dynamics affect how teams operate. For example, we could investigate if teams avoided attacking other teams that they knew were experts at capturing and reusing exploits.

Perform Temporal Study to Learn How Strategies and Competencies Have Evolved

Understanding how strategies and competencies have evolved over time could indicate the rate at which cyber security research is advancing. Therefore, an area for future work would be to compare the metrics we have from a temporal perspective and study the evolution of competencies. Intuitively, as analysis and development tools and techniques become more advanced, we can expect the metrics to reflect this. Questions we would be interested in answering include:

- Are teams finding vulnerabilities and developing exploits faster?
- Are teams patching vulnerabilities faster?
- Are teams using more sophisticated exploits?
- Are teams using more sophisticate strategies?

Perform a Team-Focused Study for Team-Specific Strategies

Understanding team-specific strategies would give us an idea of how sophisticated teams have become. Therefore, an area of future work would be to investigate whether teams have their own specific strategies. In addition, we would also want to investigate whether the strategies employed by teams differ regionally. Questions we would be interested in answering include:

- Are there certain actions that each team performs across all CTFs?
- Are these strategies dependant on who their opponents are?
- What specific strategies or actions do teams from a certain regions employ? (e.g., Do the Korean teams have different strategies from Russian teams?)
- How different are strategies employed by one team (or region) different from the others?
- How effective are these strategies?
- What if team members defect to join other teams. Would the team member bring their strategies over to the new team?

Analyze Other Attack-Defend CTFs

In order to improve how CTFs are conducted, it would be useful to also study other Attack-Defend type CTFs to learn how participating teams operate in those CTFs. Examples for other Attack-Defend type CTFs include iCTF, SSCTF and RuCTF. Questions we would like to answer include:

- How do strategies employed by the participating teams differ between each CTFs. Do they differ from those employed in DEF CON 22 CTF?
- What metrics can we measure for the other CTFs and what can we learn from these metrics?
- What interesting or surprising findings can we discover about how teams play these CTFs?

Study the Players Themselves

In addition to studying the captured network traffic and game state data, we could also learn a lot from studying the players themselves. Some areas to investigate are:

- How and with whom do the players interact with within the team, and between teams?
- Do teams have some form of organization or distribution of roles that make them effective? If so, what different roles are there?
- How homogeneous are teams and does the diversity of the team affect its game-play or performance?
- What are the average stress levels like for each team? Do more competent teams experience less stress compared to less competent teams? Are stress levels consistent throughout the CTF event or are there times that teams consider to be more stressful?

In order to get data related to the players themselves, we may have to resort to recording the teams on video, and providing pulse monitoring wristbands.

Create New Ways to Instrument the CTF to Collect the Data We Need

The data for what we want to study may not be available if the CTFs are executed in their current form. Therefore we would need to think about how we can better instrument the CTF and/or change the CTF format if necessary in order to be able to get the data we need.

THIS PAGE INTENTIONALLY LEFT BLANK

List of References

[1] L. B. Syndicate, 2016. [Online]. Available: https://media.defcon.org/DEF\%20CON\%2022/DEF\%20CON\%2022\%20ctf/DEF\%20CON\%2022\%20ctf\%20scoreboard.rar

[2] A. Dabrowski, M. Kammerstetter, E. Thamm, E. Weippl, and W. Kastner, "Leveraging competitive gamification for sustainable fun and profit in security education," in *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. Washington, D.C.: USENIX Association, Aug. 2015. [Online]. Available: http://blogs.usenix.org/conference/3gse15/summit-program/presentation/dabrowski

[3] M. Carlisle, M. Chiaramonte, and D. Caswell, "Using ctfs for an undergraduate cyber education," in *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. Washington, D.C.: USENIX Association, Aug. 2015. [Online]. Available: http://blogs.usenix.org/conference/3gse15/summit-program/presentation/carlisle

[4] T. Chothia and C. Novakovic, "An offline capture the flag-style virtual machine and an assessment of its value for cybersecurity education," in *2015 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 15)*. Washington, D.C.: USENIX Association, Aug. 2015. [Online]. Available: https://www.usenix.org/conference/3gse15/summit-program/presentation/chothia

[5] K. Chung and J. Cohen, "Learning obstacles in the capture the flag model," in *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*. San Diego, CA: USENIX Association, Aug. 2014. [Online]. Available: https://www.usenix.org/conference/3gse14/summit-program/presentation/chung

[6] A. Davis, T. Leek, M. Zhivich, K. Gwinnup, and W. Leonard, "The fun and future of ctf," in *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*. San Diego, CA: USENIX Association, Aug. 2014. [Online]. Available: https://www.usenix.org/conference/3gse14/summit-program/presentation/davis

[7] G. Vigna, K. Borgolte, J. Corbetta, A. Doupé, Y. Fratantonio, L. Invernizzi, D. Kirat, and Y. Shoshitaishvili, "Ten years of ictf: The good, the bad, and the ugly," in *2014 USENIX Summit on Gaming, Games, and Gamification in Security Education (3GSE 14)*. San Diego, CA: USENIX Association, Aug. 2014. [Online]. Available: https://www.usenix.org/conference/3gse14/summit-program/presentation/vigna

[8] O. Tsai, “Orange: Hitcon win the 2nd in defcon 22 ctf final,” 2014. [Online]. Available: http://blog.orange.tw/2014/08/hitcon-win-2nd-in-defcon-22-ctf-final.html

[9] Jeffxx.com, “Defcon 22 final hitcon - jeffxx blog,” 2014. [Online]. Available: http://www.jeffxx.com/blog/2014/08/13/2014-defcon-22-final-can-sai-xin-de-shang/

[10] Ddaa.logdown.com, “defcon 22 ctf final diaries « no ctf no life,” 2014. [Online]. Available: http://ddaa.logdown.com/posts/220500-defcon-22-ctf-diaries

[11] c. team, “Ctftime.org / all about ctf (capture the flag),” 2016. [Online]. Available: https://ctftime.org/ctf-wtf/

[12] T. Tangent, “Def con® hacking conference - about,” 2016. [Online]. Available: https://www.defcon.org/html/links/dc-about.html

[13] C. Sabrina Korber, “Cyberplayers duke it out in a world series of hacking,” 2013. [Online]. Available: http://www.cnbc.com/2013/11/08/defcon-capture-the-flag-competition-is-only-for-top-hackers.html

[14] T. Tangent, “Def con® hacking conference - ctf history,” 2016. [Online]. Available: https://www.defcon.org/html/links/dc-ctf-history.html

[15] ctftime, “Ctftime.org / ctf teams,” 2016. [Online]. Available: https://ctftime.org/stats/2014

[16] P. Pwning, “Welcome to plaid ctf 2015,” 2016. [Online]. Available: http://www.plaidctf.com/

[17] Bostonkey.party, “bkp ctf,” 2016. [Online]. Available: https://bostonkey.party/

[18] Ructf.org, “Ructfe 2014 index,” 2016. [Online]. Available: https://ructf.org/e/2014/

[19] L. B. Syndicate, 2016. [Online]. Available: https://media.defcon.org/DEF\%20CON\%2022/DEF\%20CON\%2022\%20ctf/DEF\%20CON\%2022\%20ctf\%20pre-qual\%20stat\%20dump.rar

[20] S. 0, “Def con 22 capture the flag - stratum 0,” 2014. [Online]. Available: https://stratum0.org/blog/posts/2014/08/29/defcon-ctf-2014

[21] Routards.org, “Routards team blog: Defcon 22 ctf - badger,” 2016. [Online]. Available: http://www.routards.org/2014/08/defcon-22-ctf-badger.html

[22] 2016. [Online]. Available: https://legitbs.net/2014/

[23] Tools.netsa.cert.org, “Silk — faq,” 2016. [Online]. Available: http://tools.netsa.cert.org/silk/faq.html\#what-is-flow

[24] Tools.netsa.cert.org, "Silk," 2016. [Online]. Available: https://tools.netsa.cert.org/silk/

[25] J. Ritter, "ngrep - network grep," 2016. [Online]. Available: http://ngrep.sourceforge.net/

[26] G. Developers, "Charts | google developers," 2016. [Online]. Available: https://developers.google.com/chart/

[27] J. Smith, "automayt/flowplotter," 2014. [Online]. Available: https://github.com/automayt/FlowPlotter

[28] Gephi.org, "Gephi - the open graph viz platform," 2016. [Online]. Available: https://gephi.org

[29] C. Anley and J. Koziol, *The shellcoderś handbook*. Wiley Pub., 2007.

[30] D. Kong, D. Tian, Q. Pan, P. Liu, and D. Wu, "Semantic aware attribution analysis of remote exploits," *Security and Communication Networks*, vol. 6, no. 7, pp. 818–832, 2013. [Online]. Available: http://dx.doi.org/10.1002/sec.613

[31] M. Cherukuri, S. Mukkamala, and D. Shin, "Similarity analysis of shellcodes in drive-by download attack kits," in *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2012 8th International Conference on*, Oct 2012, pp. 687–694.

[32] H. Megahed, "Linux/x86 execve /bin/sh shellcode 23 bytes," 2016. [Online]. Available: http://shell-storm.org/shellcode/files/shellcode-827.php

[33] J. Salwang, "Linux/arm - execve(/bin/sh, /bin/sh, 0) - 30 bytes," 2016. [Online]. Available: http://shell-storm.org/shellcode/files/shellcode-665.php

[34] 2016. [Online]. Available: https://media.defcon.org/DEF\%20CON\%2022/DEF\%20CON\%2022\%20ctf/DEF\%20CON\%2022\%20ctf\%20teams/

[35] Legitbs.net, "Def con ctf 2014," 2016. [Online]. Available: https://legitbs.net/2014/

[36] Pypi.python.org, "dtw 1.2 : Python package index," 2016. [Online]. Available: https://pypi.python.org/pypi/dtw/1.2

[37] Wikipedia, "Pearson product-moment correlation coefficient," 2016. [Online]. Available: https://en.wikipedia.org/wiki/Pearson_product-moment_correlation_coefficient

THIS PAGE INTENTIONALLY LEFT BLANK

Initial Distribution List

1. Defense Technical Information Center
   Ft. Belvoir, Virginia

2. Dudley Knox Library
   Naval Postgraduate School
   Monterey, California